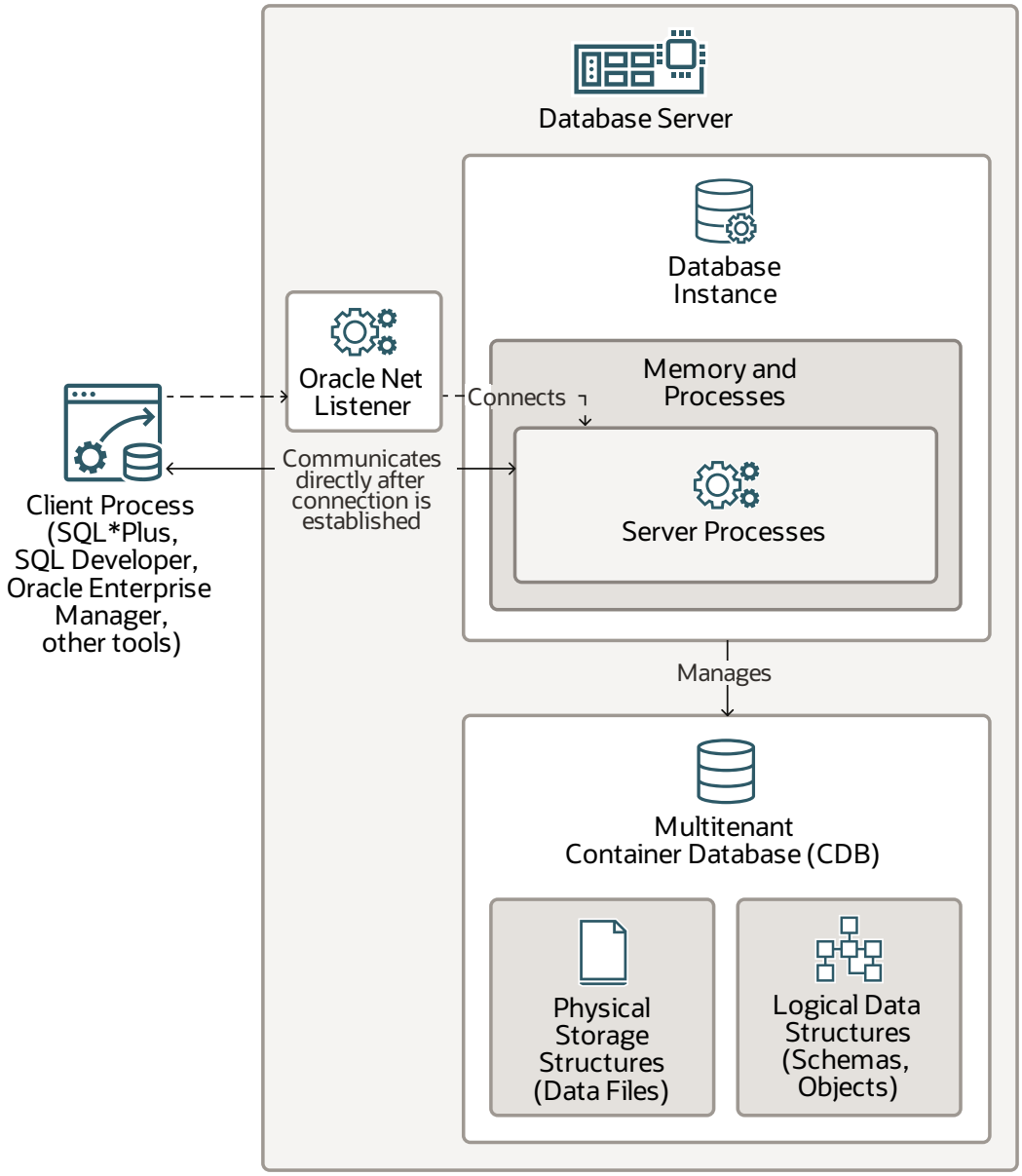
Oracle Database 23c Technical Architecture

ORACLE

September 2023

Table of Contents

Database Server



Notes

An Oracle Database consists of at least one database instance and one database. The **database instance** consists of memory and processes to manage the **multitenant container database (CDB)**. The CDB consists of physical storage structures (data files) and logical data structures (schemas and schema objects).

This diagram shows a single-instance database architecture, which has a one-to-one relationship between the database and the database instance. Multiple single-instance databases can exist on the same server machine. This configuration is useful for running 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 with a single shared database. For more information about the Oracle RAC architecture, see Oracle Real Application Clusters 19c Technical Architecture.

The **Oracle Net Listener** is a database server process that receives incoming connections from client processes, establishes connections to the database instance, and then hands over the client connections to communicate directly with the server processes to fulfill the client requests. The listener for a single-instance database can run locally on the database server or run remotely.

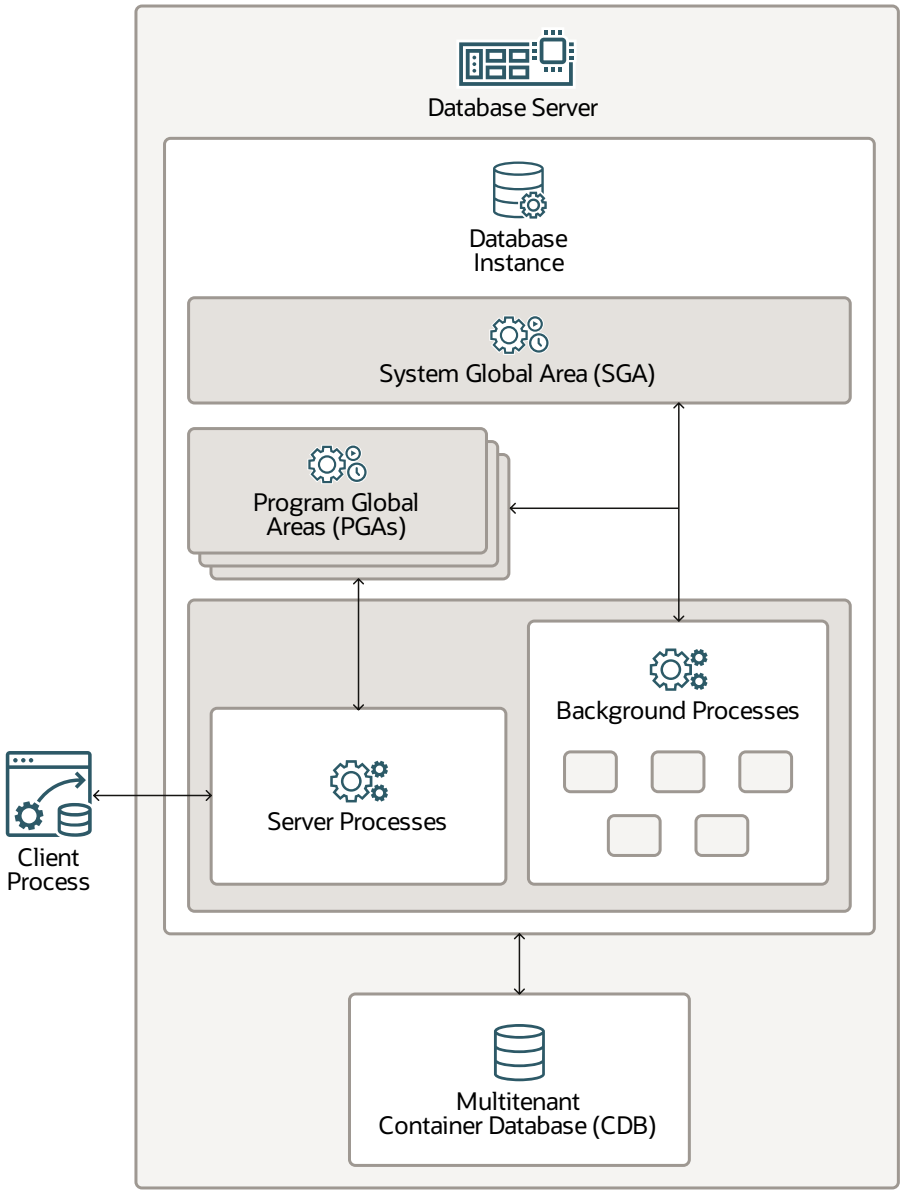
You can use several tools to administer the database, including the following:

- Oracle Data Pump
- Oracle Enterprise Manager
- Oracle Database Configuration Assistant (DBCA)
- Oracle Database Upgrade Assistant (DBUA)
- Oracle Recovery Manager (RMAN)
- Oracle Server Control Utility (SRVCTL)
- SQL Developer
- SQL*Loader
- SQL*Plus

Related Resources

- Database Architecture
- Database Instance
- Multitenant Database Architecture

Database Instance



Notes

In Oracle Database, a **database instance** is a set of processes and memory structures that manage database files. The main memory structures are the **system global area (SGA)** and the **program global areas (PGAs)**. The **background processes** operate on the database files and use the memory structures to do their work. A database instance exists only in memory.

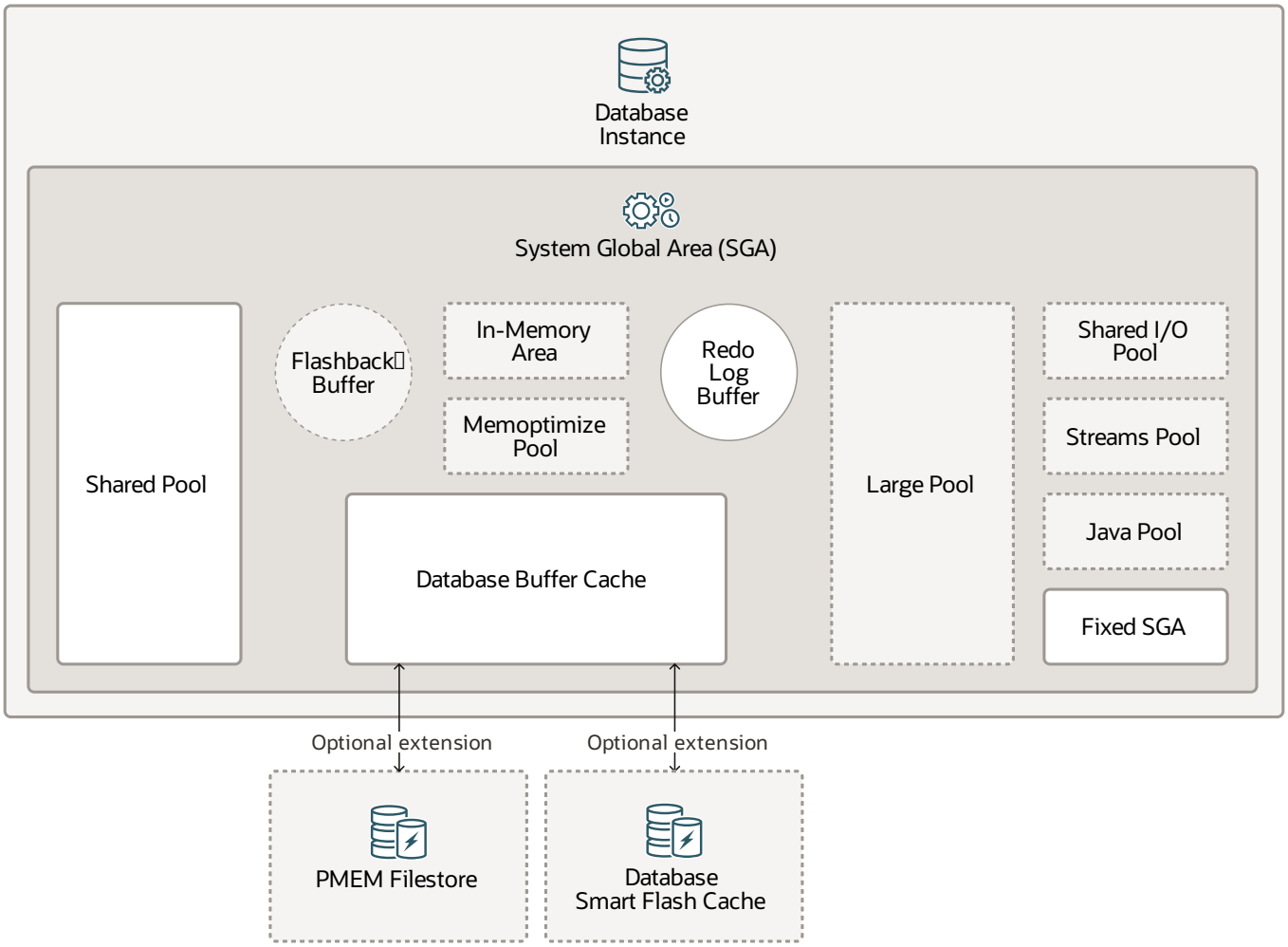
The database instance also has **server processes** to handle the connections to the database on behalf of client programs and to perform the work for the client processes. For example, these server processes parse and run SQL statements and retrieve and return results to the client processes. These types of server processes are also called foreground processes.

With the multitenant architecture, a database instance is associated with a single **multitenant container database (CDB)**. All pluggable databases (PDBs) that are plugged into the CDB (either directly or through an application container) share a single set of background processes and SGA.

Related Resources

- Database Instance
- Program Global Area (PGA)
- System Global Area (SGA)

System Global Area



Notes

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:

- The **shared pool** caches various constructs that multiple users can share; for example, the shared pool stores parsed SQL, PL/SQL code, system parameters, and data dictionary information. Almost every operation that occurs in the database involves the shared pool. For example, if a user runs a SQL statement, then Oracle Database accesses the shared pool.

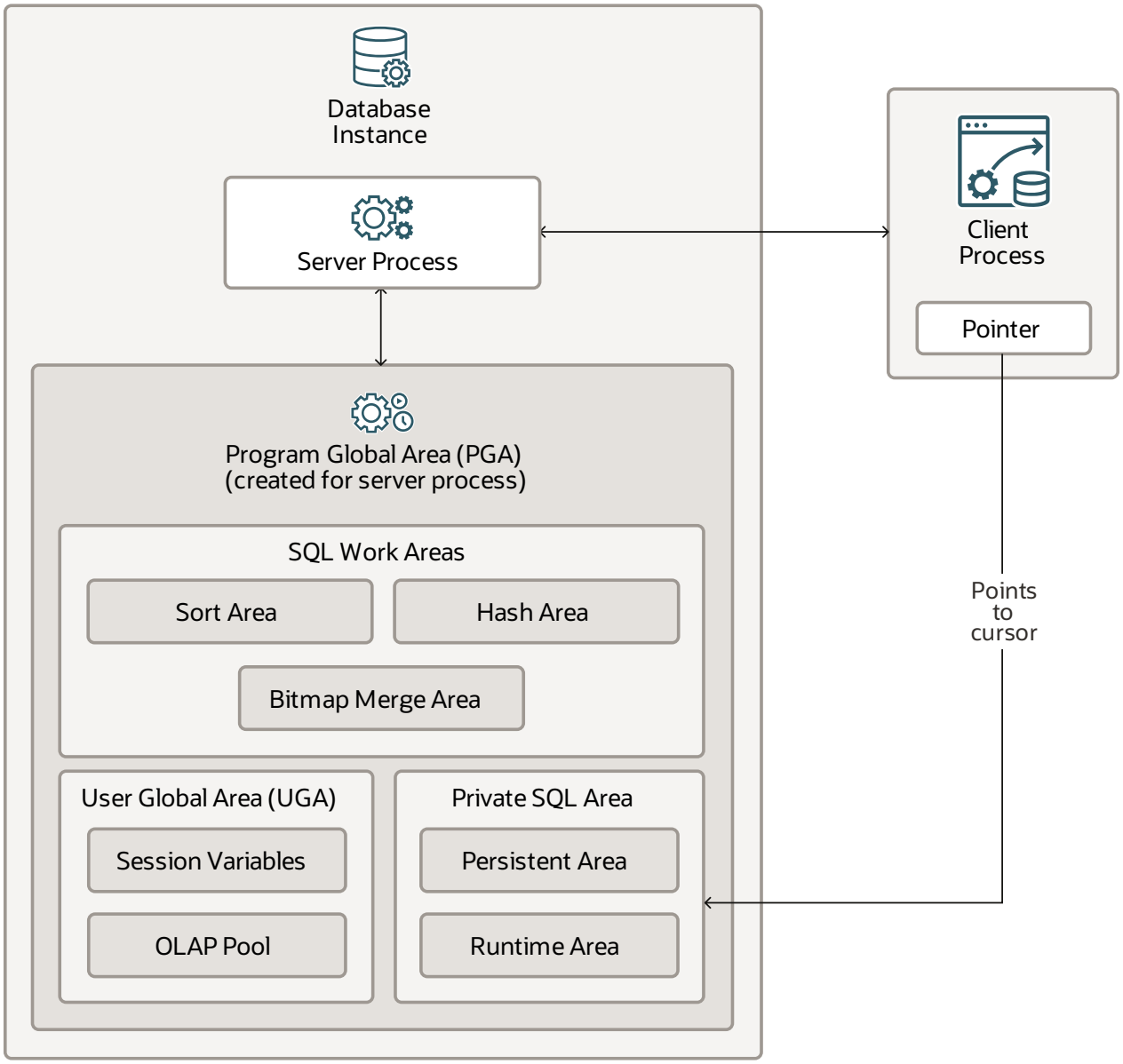
- (Optional) The **flashback buffer** works with Flashback Database so you can rewind data back in time to correct any problems. When Flashback Database is enabled, the Recovery Writer process (RVWR) starts. RVWR periodically copies modified blocks from the buffer cache to the flashback buffer, and it sequentially writes Flashback Database data from the flashback buffer to the Flashback Database logs, which are circularly reused.
- The **database buffer cache** is the memory area that stores copies of data blocks that are 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 who are concurrently connected to a database instance share access to the buffer cache.
- (Optional) If you enable the Oracle Persistent Memory Filestore (**PMEM Filestore**) and migrate data files into the PMEM Filestore, then database files are mapped for direct read-only access. Queries can bypass the traditional buffer cache mechanism, avoiding unnecessary I/O.
- (Optional) The **Database Smart Flash Cache** is a memory extension of the database buffer cache for databases that are 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 it is an order of magnitude faster than disk drives.
- The **redo log buffer** is a circular buffer in the SGA that holds information about changes to the database. This information is stored in redo entries. Redo entries contain the information that is necessary to reconstruct (or redo) changes to the database by data manipulation language (DML), data definition language (DDL), or internal operations. Redo entries are used for database recovery if necessary.
- (Optional) The **large pool** is 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 (which is used where transactions interact with multiple databases), message buffers that are used in the parallel processing of statements, buffers for Recovery Manager (RMAN) I/O workers, and deferred inserts.
- (Optional) The **In-Memory Area** enables objects (tables, partitions, and other types) to be stored in memory in 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 that return many rows rather than for OLTP, which operates on a few rows that return many columns.
- (Optional) The **memoptimize pool** 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 to provide fast access to the blocks of a given table (enabled for MEMOPTIMIZE FOR READ) that are 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 Oracle Database maintains it automatically.

- (Optional) The **shared I/O pool (SecureFiles)** is for large I/O operations on SecureFile Large Objects (LOBs). LOBs are a set of data types that hold large amounts of data. SecureFile is an LOB storage parameter that allows deduplication, encryption, and compression.
- (Optional) The **streams pool** works with Oracle Data Pump and Oracle GoldenGate processes. The streams pool stores buffered queue messages. Unless you specifically configure it, the size of the streams pool starts at zero.
- (Optional) The **Java pool** is 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.
- The **fixed SGA** is an internal housekeeping area that contains general information about the state of the database and database instance and information that is communicated between processes.

Related Resources

- Database Buffer Cache
- Fixed SGA
- In-Memory Area
- Java Pool
- Large Pool
- Redo Log Buffer
- Shared Pool
- System Global Area (SGA)

Program Global Area



Notes

The **program global area (PGA)** is a nonshared memory region within the database instance that contains data and control information exclusively for use by a server or background process. Oracle Database creates server processes to handle connections to the database on behalf of client programs. (The diagram shows a PGA for a server process.)

In a dedicated server environment, Oracle Database creates one PGA for each server and background process that starts. Oracle Database deallocates a PGA when the associated server or background process is terminated. In a dedicated server session, the PGA consists of the following components:

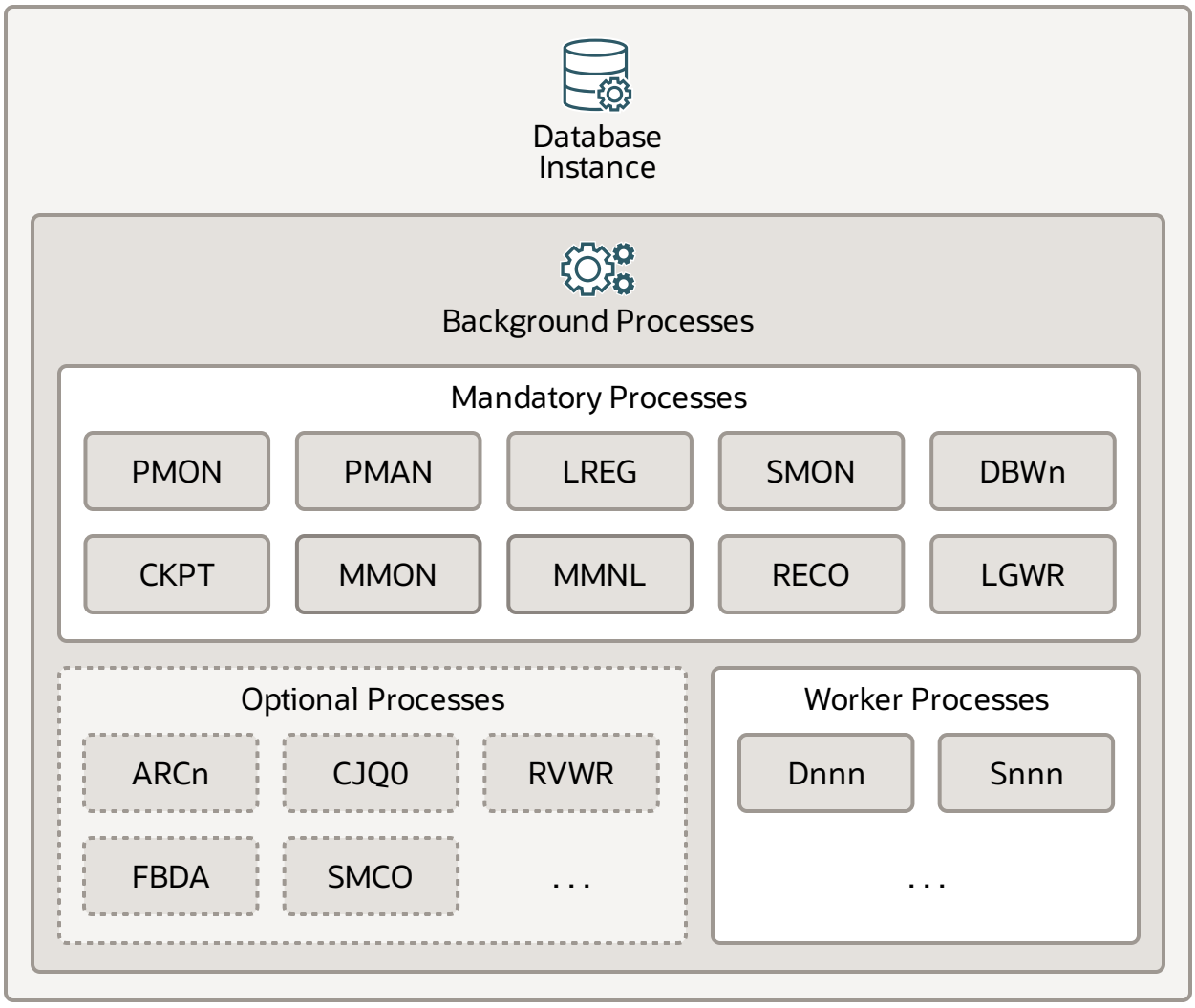
- **SQL work areas** include the sort area for functions that order data, such as ORDER BY and GROUP BY, a hash area for performing hash joins of tables, and a bitmap merge area for merging data that is retrieved from scans of multiple bitmap indexes.
- The **user global area (UGA)** is a user session data storage area for session variables, such as logon information and other information that a database session requires, and the OLAP pool that manages OLAP data pages, which are equivalent to data blocks.
- The **private SQL area** holds information about a parsed SQL statement and other session-specific information for processing. When a server process runs 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 runtime 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.

In a shared server environment, multiple client users share the server process. The UGA moves into the large pool, leaving the PGA with only SQL work areas and the private SQL area.

Related Resources

- Private SQL Area
- Program Global Area (PGA)
- SQL Work Areas
- User Global Area (UGA)

Background Processes



Notes

Background processes are part of the database instance and perform maintenance tasks 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 you're using 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 that was 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 (PMON), process manager

(PMAN), listener registration (LREG), system monitor (SMON), database writer (DBWn), checkpoint (CKPT), manageability monitor (MMON), manageability monitor lite (MMNL), recoverer (RECO), and log writer (LGWR).

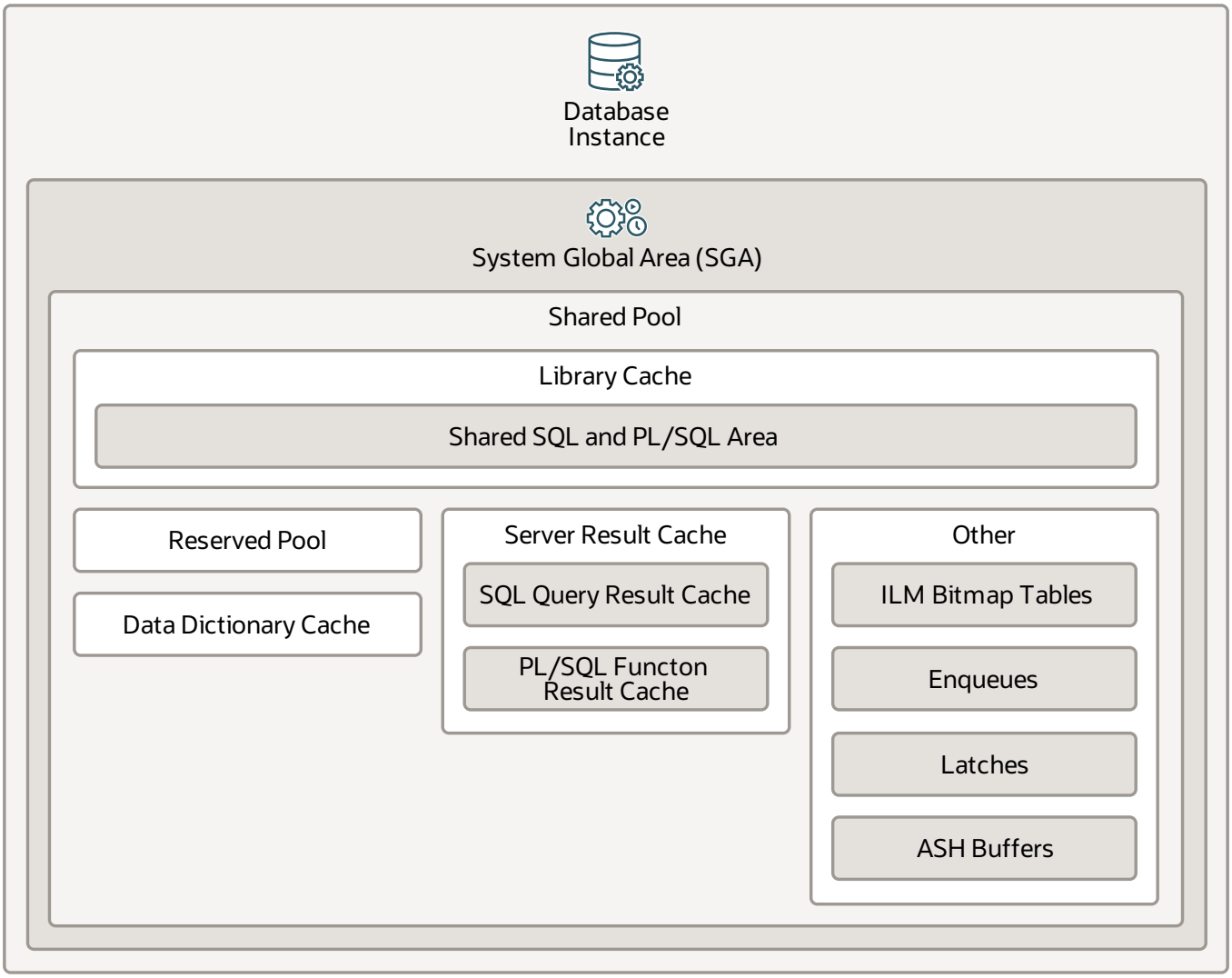
Most optional background processes are specific to tasks or features. Some common optional processes include the archiver (ARCn), job queue coordinator (CJQ0), recovery writer (RVWR), flashback data archive (FBDA), and space management coordinator (SMCO).

Worker processes are background processes that perform work on behalf of other processes. These include the dispatcher (Dnnn) and shared server (Snnn) processes.

Related Resources

- Archiver Processes (ARCn)
- Background Processes
- Checkpoint Process (CKPT)
- Database Writer Process (DBW)
- Flashback Data Archive Process (FBDA)
- Job Queue Processes (CJQ0 and Jnnn)
- Listener Registration Process (LREG)
- Log Writer Process (LGWR)
- Manageability Monitor Processes (MMON and MMNL)
- Process Architecture
- Process Manager (PMAN)
- Process Monitor Process (PMON) Group
- Recoverer Process (RECO)
- Space Management Coordinator Process (SMCO)
- System Monitor Process (SMON)

Shared Pool



Notes

The **shared pool** is a component of the system global area (SGA) within the database instance. It's 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. Almost every operation that occurs in the database involves the shared pool. For example, if a user runs a SQL statement, then Oracle Database accesses the shared pool.

The shared pool consists of the following subcomponents:

- The **library cache** is a shared pool memory structure that stores executable SQL and PL/SQL

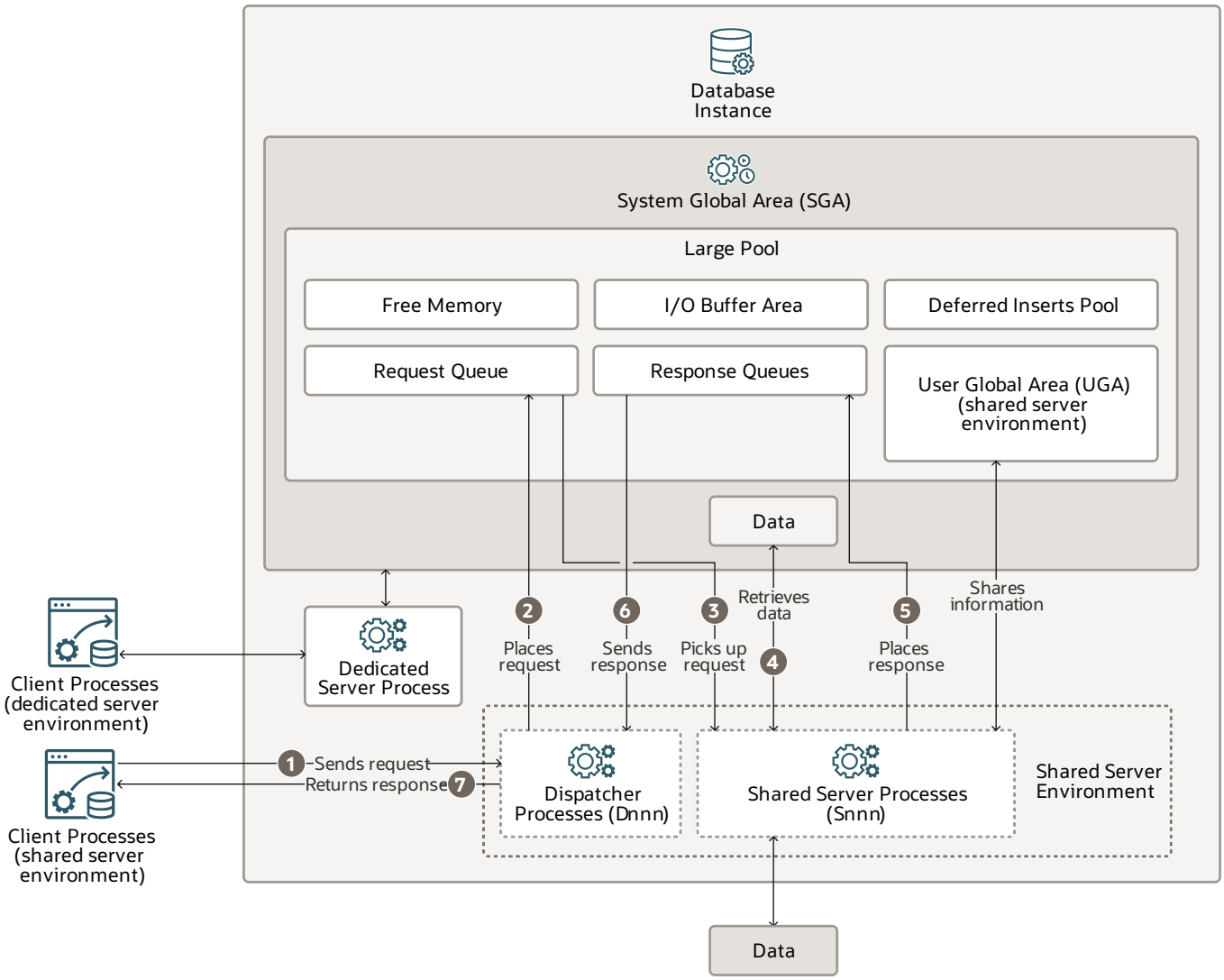
code. This cache contains the shared SQL and PL/SQL areas and control structures, such as locks and library cache handles. When a SQL statement is executed, the database attempts to reuse previously executed code. If a parsed representation of a SQL statement exists in the library cache and can be shared, 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.

- The **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.
- The **data dictionary cache** stores information about database objects (that is, dictionary data). This cache is also called the row cache because it holds data as rows instead of buffers.
- The **server result cache** is a memory pool within the shared pool that 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 to protect shared data structures in the SGA from simultaneous access. Examples are row cache objects, library cache pin, and log file parallel write.

Related Resources

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

Large Pool



Notes

The **large pool** is an optional memory area within the database instance and system global area (SGA). You can configure the large pool to provide large memory allocations for the following areas:

- The **user global area (UGA)** (session memory) for the shared server environment and the Oracle XA interface (which is used where transactions interact with multiple databases). In a dedicated server environment, the UGA is stored in the program global area (PGA).
- The **I/O buffer area** includes I/O server processes, message buffers for parallel query operations, buffers for Recovery Manager (RMAN) I/O workers, and advanced queuing memory table storage.

- The **deferred inserts pool** is used by the fast ingest feature, which enables high-frequency, single-row data inserts into the database for tables that are defined as MEMOPTIMIZE FOR WRITE. The inserts by fast ingest are also called deferred inserts. They are initially buffered in the large pool and later written to disk asynchronously by the Space Management Coordinator process (SMCO) and Wxxx worker background processes after 1MB worth of writes per session per object (or after 60 seconds). Sessions can't read any data that is buffered in this pool, even when committed, 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 that is 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 dedicated 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:

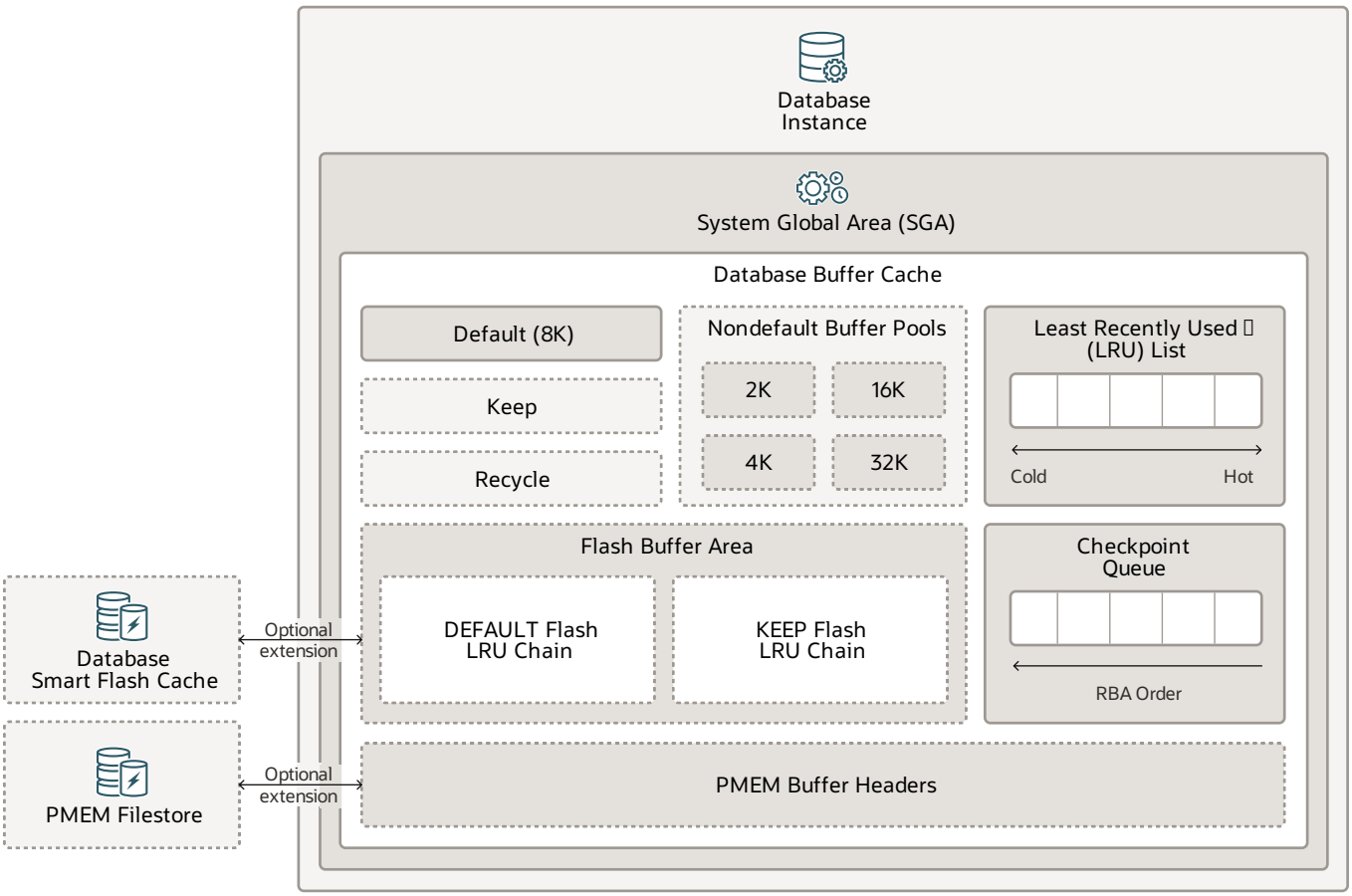
1. A client process sends a request to the database instance, and a **dispatcher process (Dnnn)** receives that request.
2. The dispatcher places the request in the **request queue** in the large pool.
3. The next available **shared server process (Snnn)** picks up the request. The shared server processes 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 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 process.

Related Resources

- Dedicated Server Architecture
- Large Pool
- User Global Area (UGA)
- Shared Server Architecture

Database Buffer Cache



Notes

The **database buffer cache**, also called the buffer cache, is a memory area in the system global area (SGA) of the database instance. It stores copies of data blocks that are 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 that are concurrently connected to a database instance share access to the buffer cache. The goals of the buffer cache are to optimize physical I/O, keep frequently accessed blocks in the buffer cache, and manage buffer headers that point to data files in the optional Oracle Persistent Memory Filestore (PMEM Filestore).

The first time an Oracle Database client 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 copies 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 areas:

- The **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.
- (Optional) The **keep pool** is for blocks that are expected to be accessed frequently and would have 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 to avoid I/O operations. Tables are assigned to the keep pool. They don't move from the default pool to the keep pool automatically.
- (Optional) The **recycle pool** is for blocks that are used infrequently. A recycle pool prevents objects from consuming unnecessary space in the cache.
- (Optional) **Nondefault 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.
- (Optional) **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 storing 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 (RAC) environment.
- The **least recently used (LRU) list** contains pointers to dirty and non-dirty buffers. The LRU list has a hot end and a 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.
- The **checkpoint queue** is a list of dirty buffers in the order they were changed based on the redo block address (RBA) order.
- (Optional) If you enable Database Smart Flash Cache, **the 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 is first read from disk into memory (physical read). When the in-memory buffer cache gets full, a buffer is evicted out of the memory based on an LRU mechanism. With Database Smart Flash Cache, when a clean in-memory buffer ages out, the database writer process

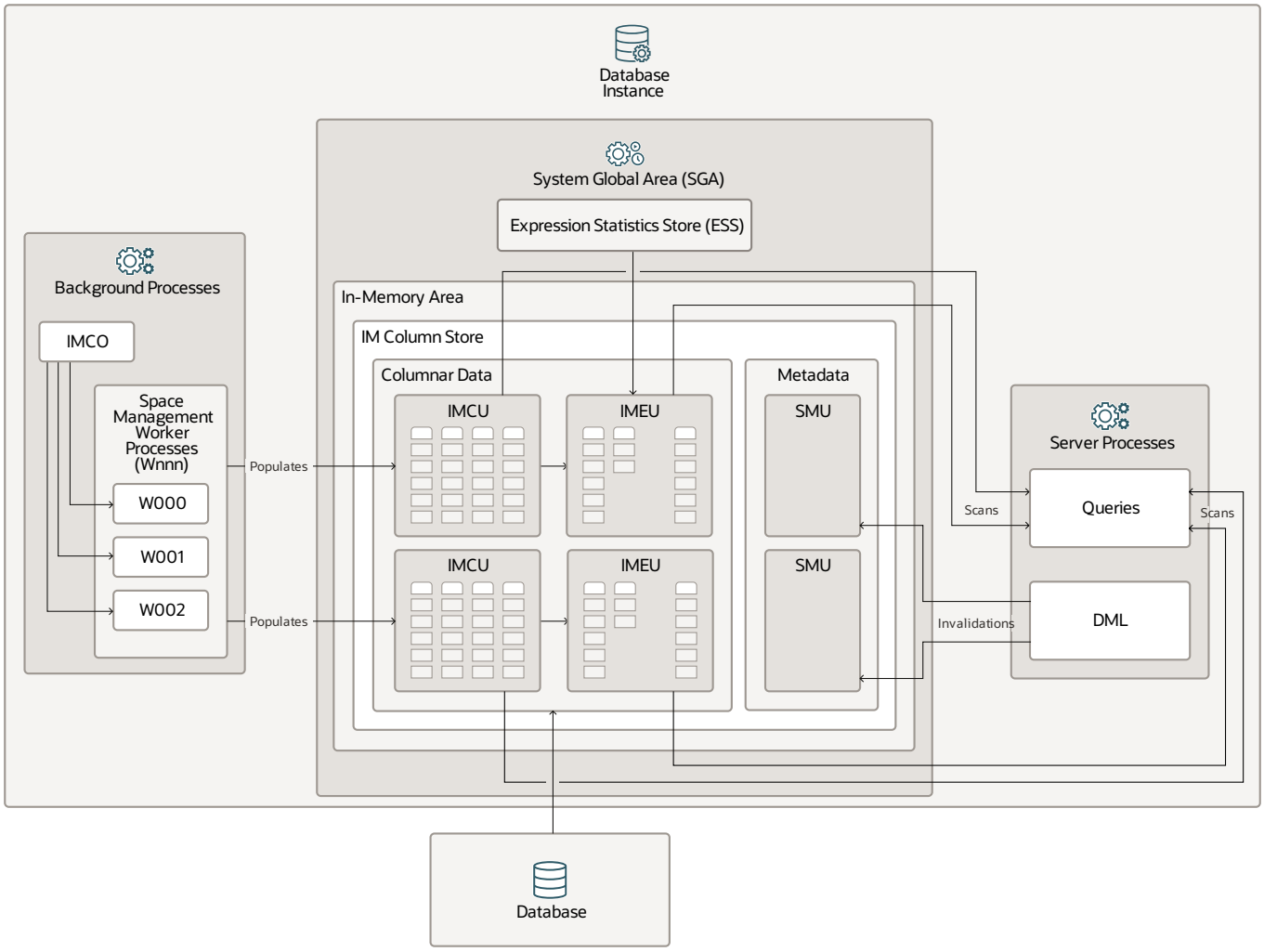
(DBWn) writes the content to the flash cache in the background, and the buffer header remains in memory as metadata in either the DEFAULT or KEEP flash LRU list, depending on the value of the FLASH_CACHE object attribute. The KEEP flash LRU list maintains the buffer headers on a separate list to prevent the regular buffer headers from replacing them. This means that the flash buffer headers belonging to an object that is 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 RAC is maintained in the same way as by Oracle RAC 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 to checkpoint them because writing to flash cache does not count for checkpoint.

- (Optional) If you enable the **PMEM Filestore** and migrate data files into the PMEM Filestore, then database files are mapped for direct read-only access. Queries can bypass the traditional buffer cache mechanism, avoiding unnecessary I/O. In this case, buffer headers must store metadata corresponding to the PMEM blocks. The database can still use the traditional (DRAM) buffer cache for modifications, read consistency, and faster access for "hot" data blocks.

Related Resources

- Database Buffer Cache

In-Memory Area



Notes

The **In-Memory Area** is an optional system global area (SGA) component within the database instance. It contains the **In-Memory column store (IM column store)**, which stores tables and partitions in memory by using a columnar format that is 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 you create or alter it. You can specify this attribute on a tablespace (for all new tables and views in the tablespace), table, partition, subpartition, 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 that is 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 might contain rows 1-1000 for the column weekly_salary. For the rows that are 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 subdivided into two pools:

- A 1MB columnar data pool that stores the actual column-formatted data that is populated into memory (IMCUs and IMEUs)
- A 64K metadata pool that stores metadata about the objects that are populated into the IM column store (SMUs)

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.

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

An in-memory table gets IMCUs allocated in the IM column store when the table data is first accessed 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 because 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 can 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 IM column store. This improves 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 IM 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 i**n-memory coordinator process (IMCO)** initiates the background population and repopulation of columnar data. The **space management coordinator process (SMCO)** and **space management worker processes (Wnnn)** 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.

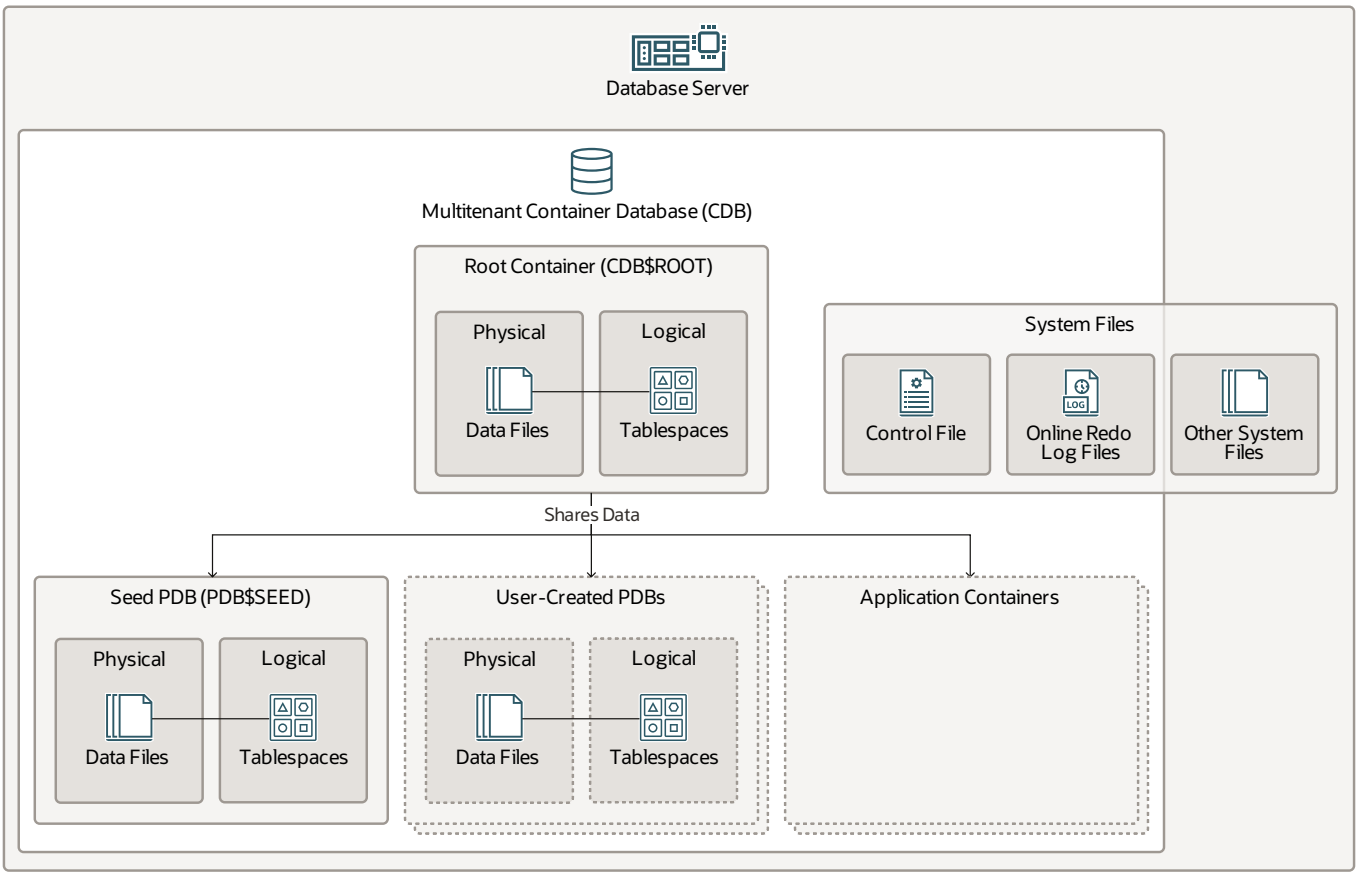
In-Memory deep vectorization can optimize complex SQL operators by pipelining the physical operators inside each SQL operator and vectorizing them using single instruction, multiple data (SIMD) techniques. This feature is enabled by default but you can disable it by setting the INMEMORY_DEEP_VECTORIZATION initialization parameter to false.

Related Resources

- Expression Statistics Store (ESS)

- In-Memory Area
- In-Memory Column Store
- In-Memory Process Architecture
- In-Memory Storage Units

Multitenant Container Database (CDB)



Notes

In a **multitenant container database (CDB)**, a container is a collection of schemas, objects, and related structures. A **pluggable database (PDB)** is a user-created container that stores the data and code for an application, such as a human resources application.

From the perspective of a user or application, the PDB appears to be a logically separate, independent database. From the operating system perspective, the CDB is the database.

Every CDB has the following containers:

- One **root container**, called CDB$ROOT, that stores Oracle-supplied metadata and common users (database users that are known in every container belonging to that CDB). The root does not store any user data. All PDBs belong to the root.
- One **seed PDB**, called PDB$SEED, which is a system-supplied template that the CDB can use to create new PDBs. You can't modify the seed PDB.
- Zero or more **user-created PDBs**. No PDBs exist when you first create a CDB. You can plug multiple PDBs into a CDB at one time, but you can't plug a PDB into multiple CDBs. Each PDB exists in complete isolation from the other PDBs that are plugged into the same CDB. Users interact only with the PDBs and not the seed PDB or root container.
- Zero or more **application containers**, which are optional collections of PDBs within a CDB that store data and metadata for applications. An application container, like the CDB, can include multiple application PDBs, and enables these PDBs to share metadata and data.

At a physical level, containers consist of physical **data files**. Each container has at least one data file. At a logical level, the database allocates data across data files with logical structures called **tablespaces**.

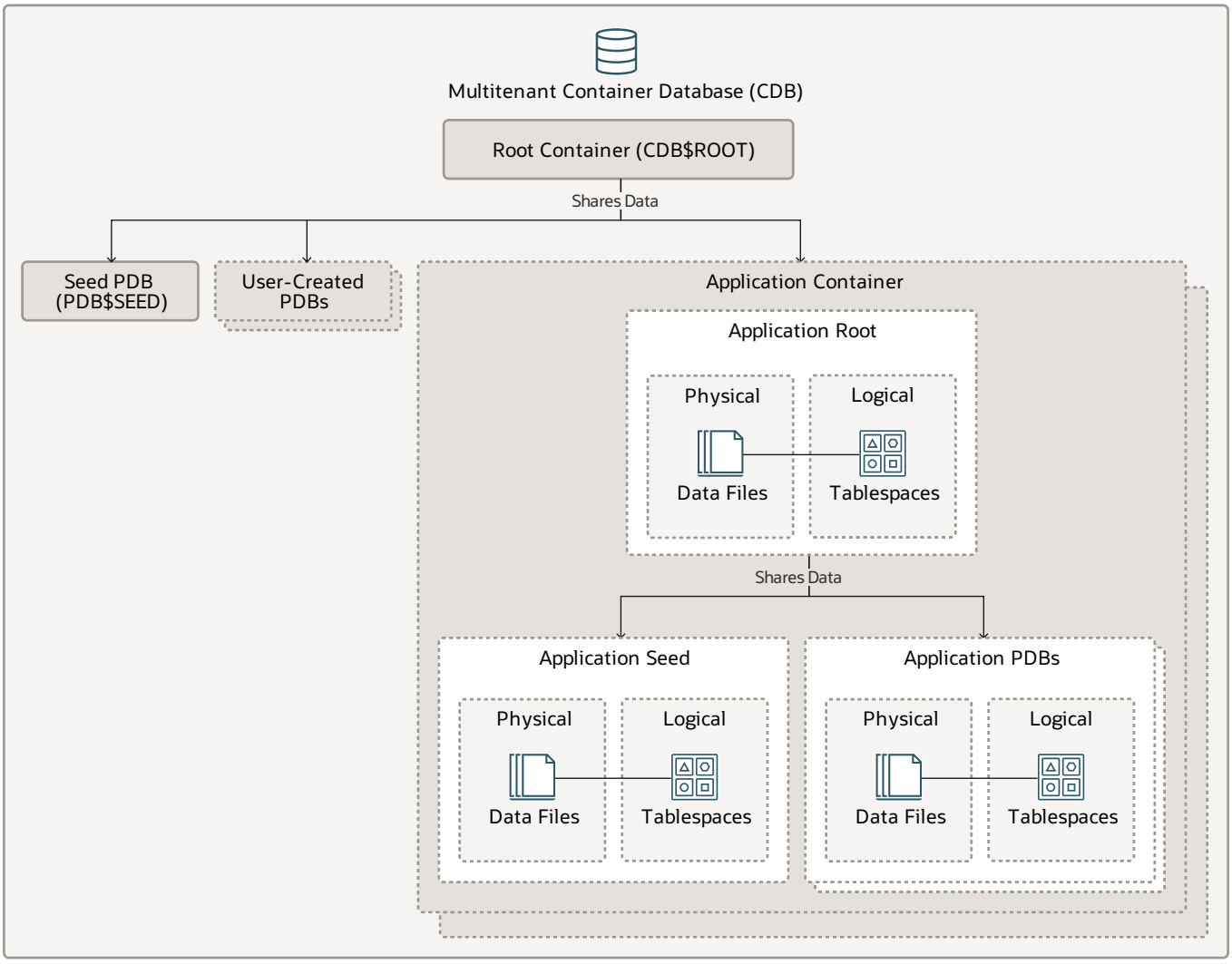
The CDB also uses several system files during its operation. The control file and redo online log files reside in the CDB and are shared across all PDBs and application containers. Other system files reside outside the CDB.

Note: Non-CDBs were desupported in Oracle Database 21c, which means that the Oracle Universal Installer and Database Configuration Assistant (DBCA) can no longer create non-CDB Oracle Database instances.

Related Resources

- Application Containers
- Container Databases (CDBs)
- Logical Storage Structures
- Physical Storage Structures
- Pluggable Databases (PDBs)

Application Containers



Notes

An **application container** is an optional, user-created component within a **multitenant container database (CDB)**. An application container stores data and metadata for pluggable databases (PDBs) that are specific to an application.

In some ways, an application container functions as an application-specific CDB within a CDB. An application container, like the CDB itself, can include multiple **application PDBs** and enables these PDBs to share data and metadata in the same way that the **CDB root** shares data and metadata with the seed PDB, user-created PDBs, and application containers that are plugged into the CDB root.

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 application root. The application root belongs to the CDB root. An application root differs from both the CDB root and standard PDB because it can store user-created common objects, which are accessible to the application PDBs that are plugged in to the application root.

An **application seed** is an optional, user-created PDB within an application container. An application container can have zero or one application seed. An application seed enables you to create application PDBs quickly. It serves the same role within the application container as the seed PDB serves within the CDB itself.

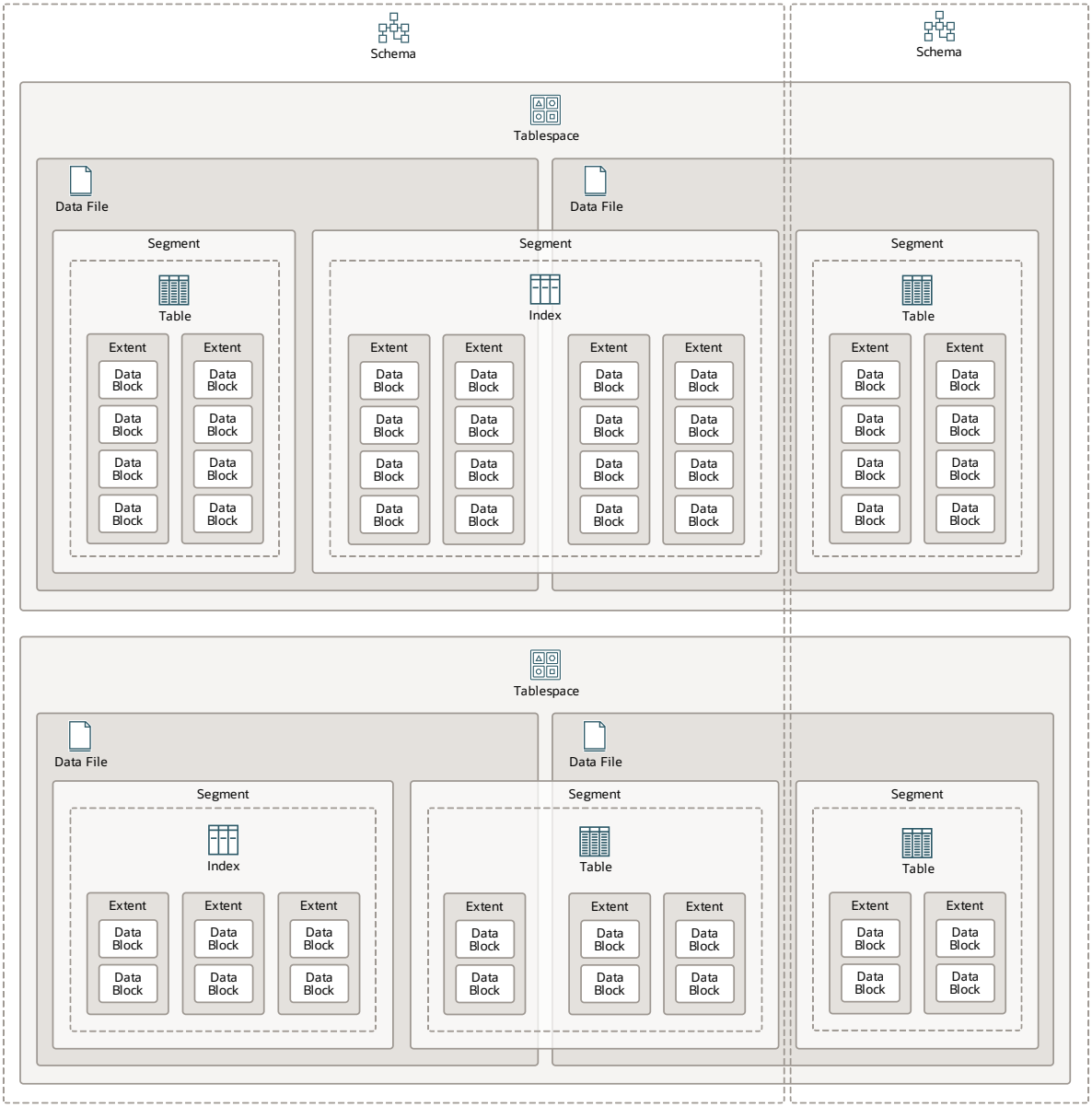
At a physical level, containers consist of physical **data files**. Each container has at least one data file. At a logical level, the database allocates data across data files with logical structures called **tablespaces**.

A typical application installs application common users, metadata-linked common objects, and data-linked common objects. For example, 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.

Related Resources

- Application Containers
- Application PDBs
- Application Root
- Application Seed

Database Storage Structures



Notes

A multitenant container database (CDB) is a collection of **schemas** and schema objects, such as **tables** and **indexes.**

At the physical level, a CDB stores objects in **data files**. You can use the following mechanisms to store data files:

- Operating system file system, which is a structure for storing and retrieving data. Most Oracle databases store files in a file system.

- Cluster file system, which is a distributed file system that is a cluster of servers that collaborate to provide consistency and high performance to their clients. In an Oracle Real Application Cluster (Oracle RAC) environment, a cluster file system makes shared storage appear as a single file system that many computers share in a clustered environment.
- Oracle Persistent Memory Filestore (PMEM Filestore), which stores database files in PMEM (byte-addressable persistent memory).
- Oracle Automatic Storage Management (Oracle ASM).

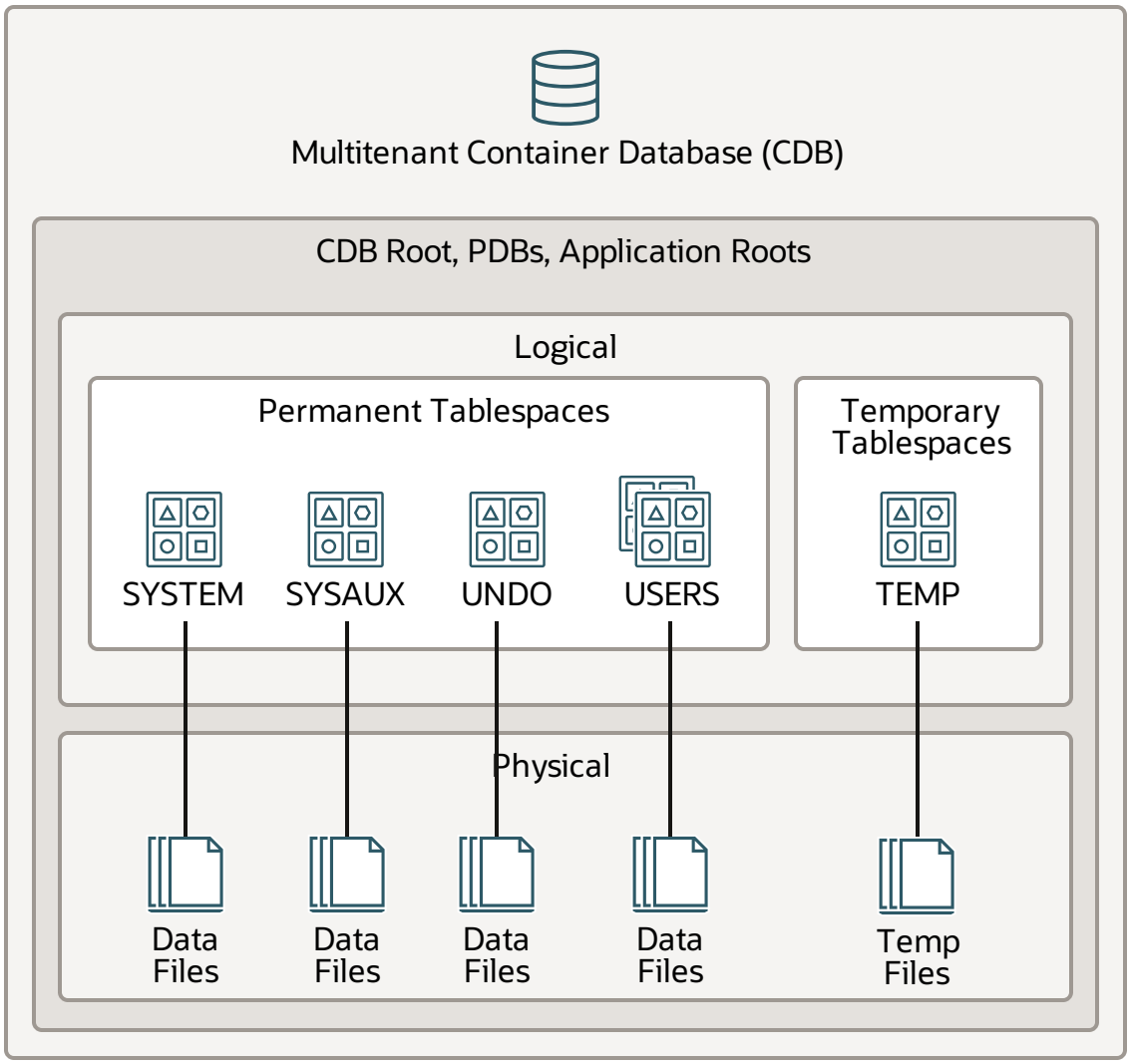
At the logical level, a CDB allocates data across the data files with the following structures:

- **Tablespaces** contain zero or more segments, which correspond to schema objects. A tablespace can contain segments that are allocated to different schemas, and the segments for one schema can span multiple tablespaces. At the physical level, a tablespace can store data in multiple data files, but multiple tablespaces cannot share data files. Each pluggable database (PDB) and application container has its own set of tablespaces. You can store user data in the default USERS tablespace or create additional tablespaces for specific application data or system data.
- **Segments** are sets of extents that are allocated for specific schema objects. Each schema object (or object partition) that consumes storage has its own segment. A segment can include extents from multiple data files, but it cannot span multiple tablespaces.
- **Extents** are sets of logically contiguous data blocks. Each extent is allocated to a schema object within a single data file. By default, Oracle Database allocates an initial extent when you create a schema object and segment. If the initial extent becomes full, the database automatically allocates another extent for the segment. The new extent does not need to be in the same data file.
- **Data blocks** are the smallest logical units of data storage. One logical data block corresponds to a specific number of bytes of persistent storage (for example, 2 KB) within a data file.

Related Resources

- Logical Storage Structures
- Physical Storage Structures
- Schema Objects

Tablespaces



Notes

A **tablespace** is a logical storage container for segments. Segments are database objects, such as tables and indexes, that consume storage space.

In a multitenant container database (CDB), each pluggable database (PDB) and application root has its own set of permanent tablespaces, which correspond to physical data files:

- One **SYSTEM tablespace**, which includes the data dictionary; tables and views that contain administrative information about the database; and compiled stored objects such as triggers, procedures, and packages.
- One **SYSAUX tablespace**, which is an auxiliary tablespace to the SYSTEM tablespace. It is the default tablespace for many Oracle Database features and products that previously required their own tablespaces, so it reduces the number of tablespaces required by the database.

- One or more **undo tablespaces**. In local undo mode (which is the default), the database automatically creates an undo tablespace in every PDB. In shared undo mode, a single-instance CDB has only one active undo tablespace exists, and an Oracle Real Application Clusters (Oracle RAC) CDB nas one active undo tablespace for every instance. All undo tablespaces in shared undo mode are visible in the data dictionaries and related views of all containers.

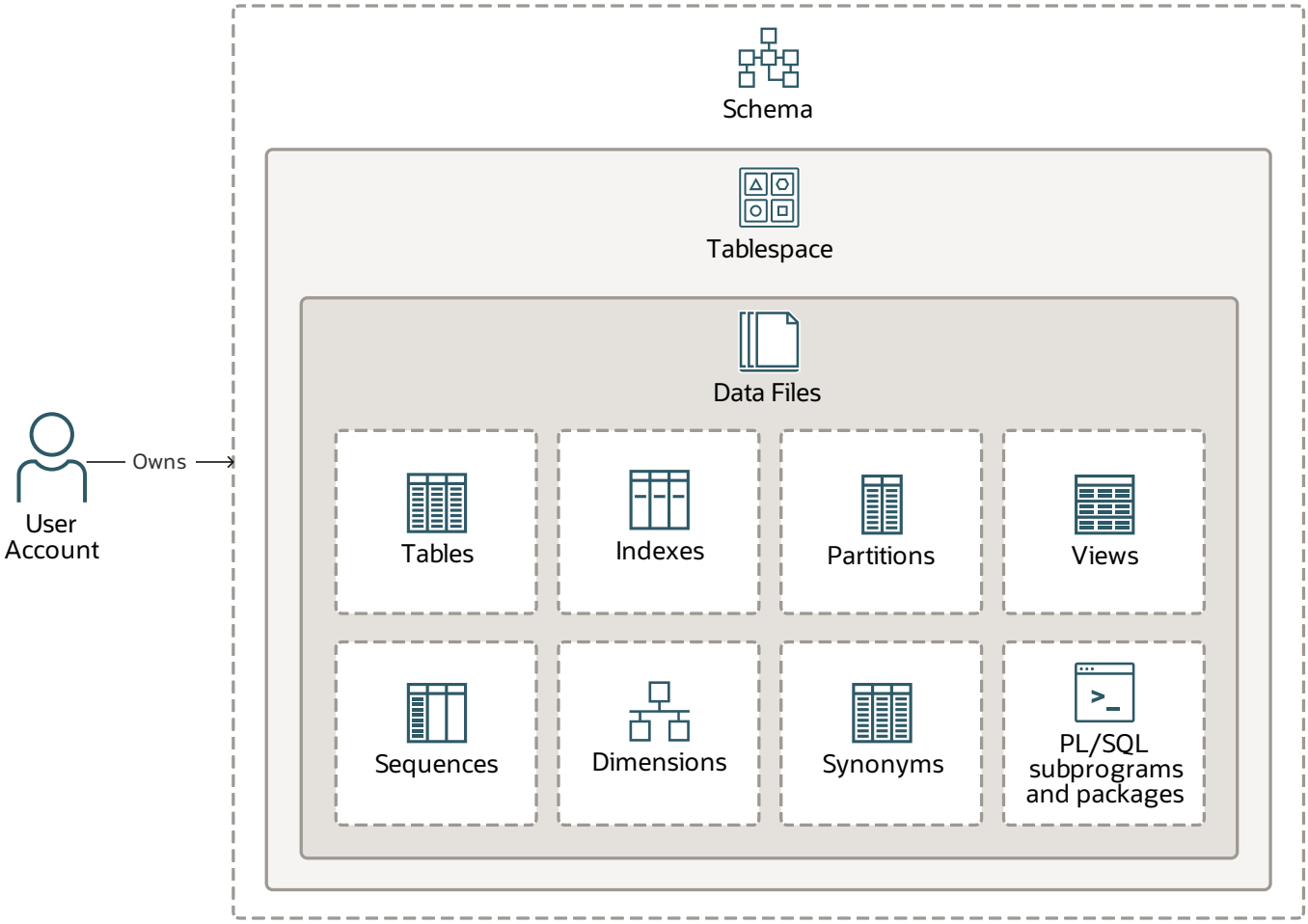
- Zero or more **user-created tablespaces** , which contain the data for user-defined schemas and objects in the PDB. You can store user data in the default USERS tablespace or create additional tablespaces for specific application data or system data.

Each container also has a **temporary tablespace**, which correspond to physical temp files. One default temporary tablespace exists for the CDB root and for each application root, application PDB, and PDB. Temporary tablespaces contain transient data that persists only for the duration of a session. No permanent schema objects can reside in a temporary tablespace.

Related Resources

- Permanent Tablespaces
- SYSAUX Tablespace
- SYSTEM Tablespace
- Tablespaces in a Multitenant Environment
- Temporary Tablespaces
- Undo Tablespaces

Schemas and Schema Objects



Notes

A database **schema** is a logical container for data structures, called schema objects. Each Oracle Database **user account** owns a single schema, which has the same name as the user.

Note: The database also stores other types of objects that are not contained in a schema. These objects include database user account, roles, contexts, and dictionary objects.

At the physical level, the database stores schema objects in data files. At the logical level, the database allocates data across the data files with tablespaces. There is no relationship between schemas and tablespaces: a tablespace can contain objects from different schemas, and the objects for a schema can be contained in different tablespaces. The data of each object is physically

contained in one or more data files.

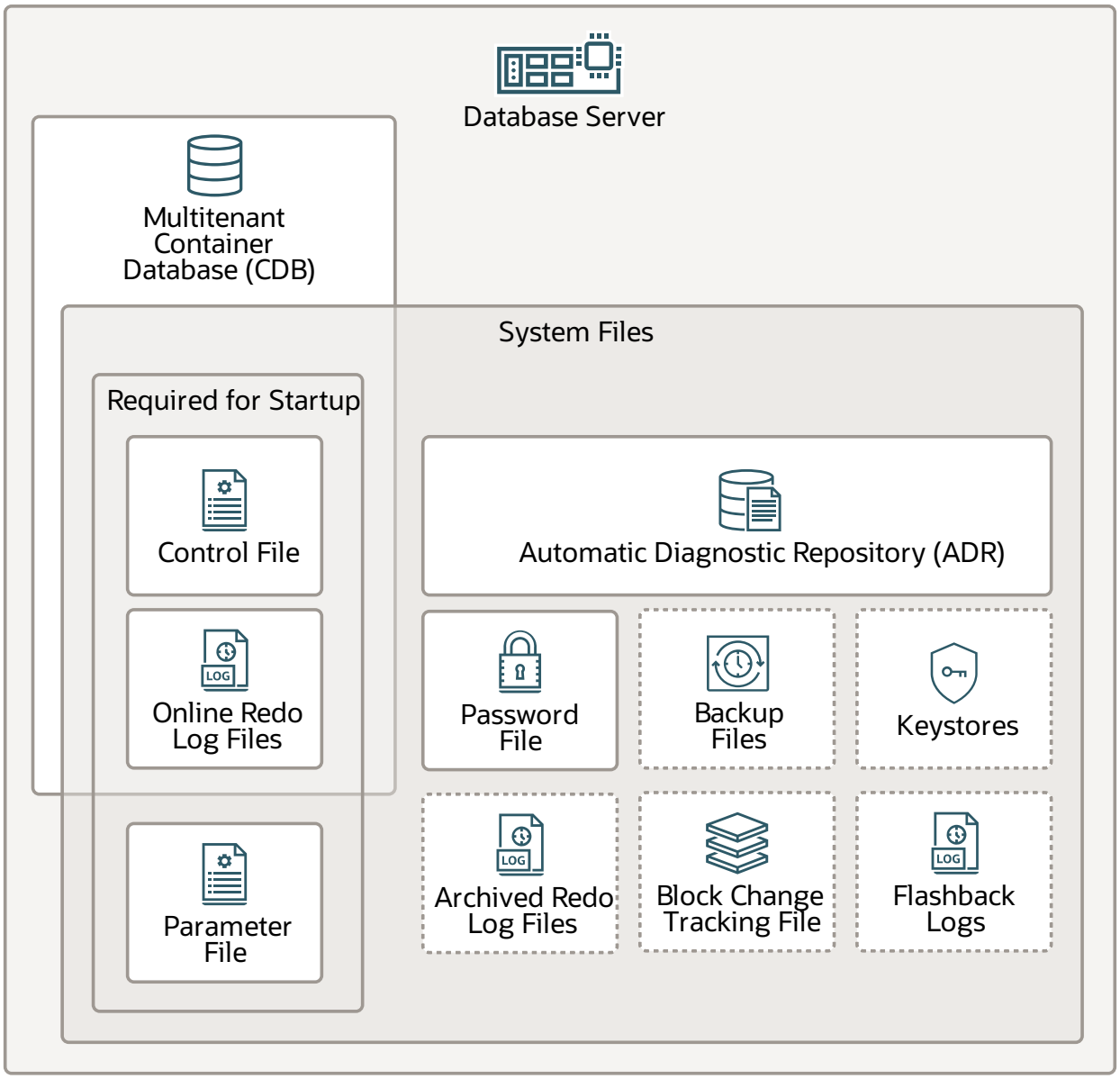
Schemas include the following principal types of objects:

- **Tables** store data in rows. Tables are the most important schema objects in a relational database.
- **Indexes** contain an entry for each indexed row of the table or table cluster and provide direct, fast access to rows.
- **Partitions** are pieces of large tables and indexes. Each partition has its own name and may optionally have its own storage characteristics.
- **Views** are customized presentations of data in one or more tables or other views. You can think of them as stored queries. Views do not actually contain data.
- **Sequences** can be shared by multiple users to generate integers. Typically, you use sequences to generate primary key values.
- **Dimensions** define parent-child relationships between pairs of column sets, where all the columns of a column set must come from the same table. However, columns in one column set (called a level) can come from a different table than columns in another set. You might use dimensions to categorize data such as customers, products, and time.
- **Synonyms** are aliases for other schema objects. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.
- PL/SQL is the Oracle procedural extension of SQL. A **PL/SQL subprogram** is a named PL/SQL block that you can invoke with a set of parameters. A PL/SQL package groups logically related PL/SQL types, variables, and subprograms.

Related Resources

- Dimensions
- Indexes
- Logical Storage Structures
- Partitions
- PL/SQL
- Schema Objects
- Sequences
- Synonyms
- Tables
- User Accounts
- Views

Database System Files



Notes

Oracle Database uses several database system files that reside on the database server. These are different from data files, which are physical files that belong to database containers.

The following files are required for database startup:

- **Control files** store metadata about the data files and online redo log files (for example, their names and statuses). The database instance requires this information to open the database. Control files also contain metadata that must be accessible when the database is not open. Each multitenant container database (CDB) has one unique control file, although multiple

identical copies are permitted. Pluggable databases (PDBs) do not have separate control files.
- The **parameter file** defines how the database instance is configured when it starts up. You can use an initialization parameter file (pfile) or a server parameter file (spfile).
- **Online redo log files** store changes to the database as they occur and are used for data recovery. These files are part of the CDB.

Note: Oracle recommends that you maintain multiple copies of the control files and online redo log files in separate locations to avoid a single point of failure. This is called multiplexing. For details, see Multiple Control Files and Multiple Copies of Online Redo Log Files

.

The database also uses the following system files, which reside outside the CDB:

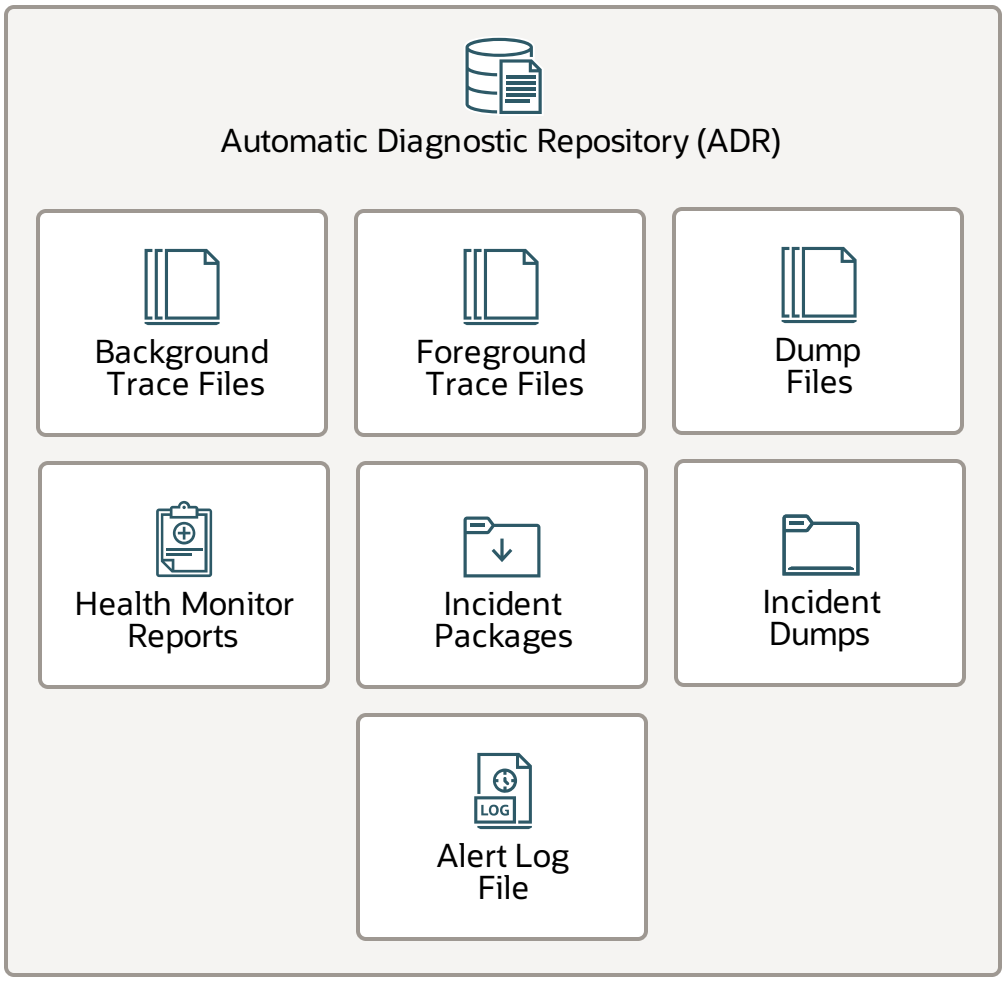
- **Automatic Diagnostic Repository (ADR)** is a file-based repository for database diagnostic data, such as traces, dumps, the alert log, health monitor reports, and more. It has a unified directory structure across multiple instances and multiple products. The database, Oracle Automatic Storage Management (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 in its own home directory within the ADR.
- (Optional) **Backup files** are used for database recovery. You typically restore a backup file when a media failure or user error has damaged or deleted the original file.
- (Optional) **Archived redo log files** contain an ongoing history of the data changes that the database instance generates. You can use these files and a backup of the database to recover lost data files.
- (Optional) The password file enables users with the SYSDBA, SYSOPER, SYSBACKUP, SYSDG, SYSKM, SYSRAC, and SYSASM roles to connect remotely to the database instance and perform administrative tasks.
- (Optional) **Keystores** (previously called wallets) are secure software containers that store authentication and signing credentials. Oracle Database supports software keystores, Oracle Key Vault, and other PKCS#11 compatible key management devices.
- (Optional) **Block change tracking file** contains changed blocks to improve the performance of incremental backups. 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.
- (Optional) Flashback Database uses **flashback logs** to access past versions of data blocks and some information from archived redo logs. This enables you to return a database to its state at a time in the recent past. Flashback Database requires that you configure a fast recovery area for a database because the flashback logs can only be stored there. You can use Flashback Database only if flashback logs are available, so you must set up your database in advance to create flashback logs.

Related Resources

- Automatic Diagnostic Repository (ADR)

- Backup and Recovery
- Control Files
- Flashback Database
- Keystores
- Online Redo Log
- Parameter Files
- Passwords for Administrative Users
- Recovery Manager (RMAN)

Automatic Diagnostic Repository (ADR)



Notes

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

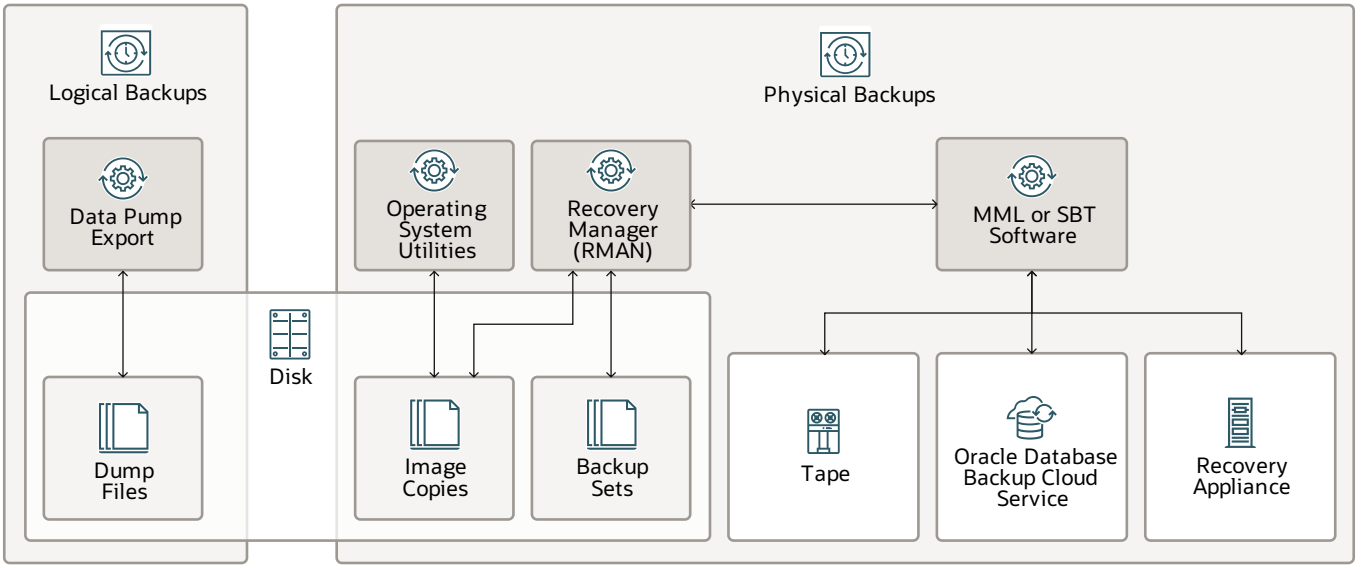
- **Background trace files** store information from database background processes. When a process detects an internal error, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, whereas other information 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** store information from server processes. 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** are 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** include results of diagnostic checks from the Health Monitor framework. 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** contain information for uploading diagnostic data to Oracle Support. 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** contain diagnostic data that Oracle Database collects 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** is a chronological log of messages and errors. Oracle recommends that you review the alert log periodically.

Related Resources

- Automatic Diagnostic Repository

Backup Files



Notes

You can create the following types of backups:

- **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 on disk. Logical backups can supplement physical backups.
- **Physical backups** are copies of physical database files. You can make physical backups with **Recovery Manager (RMAN)** or operating system utilities.

When you use RMAN, you can store the following types of backup formats on disk:

- 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 disks because you can update them incrementally and recover them in place.
- A **backup set** is a proprietary format that RMAN creates. It 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

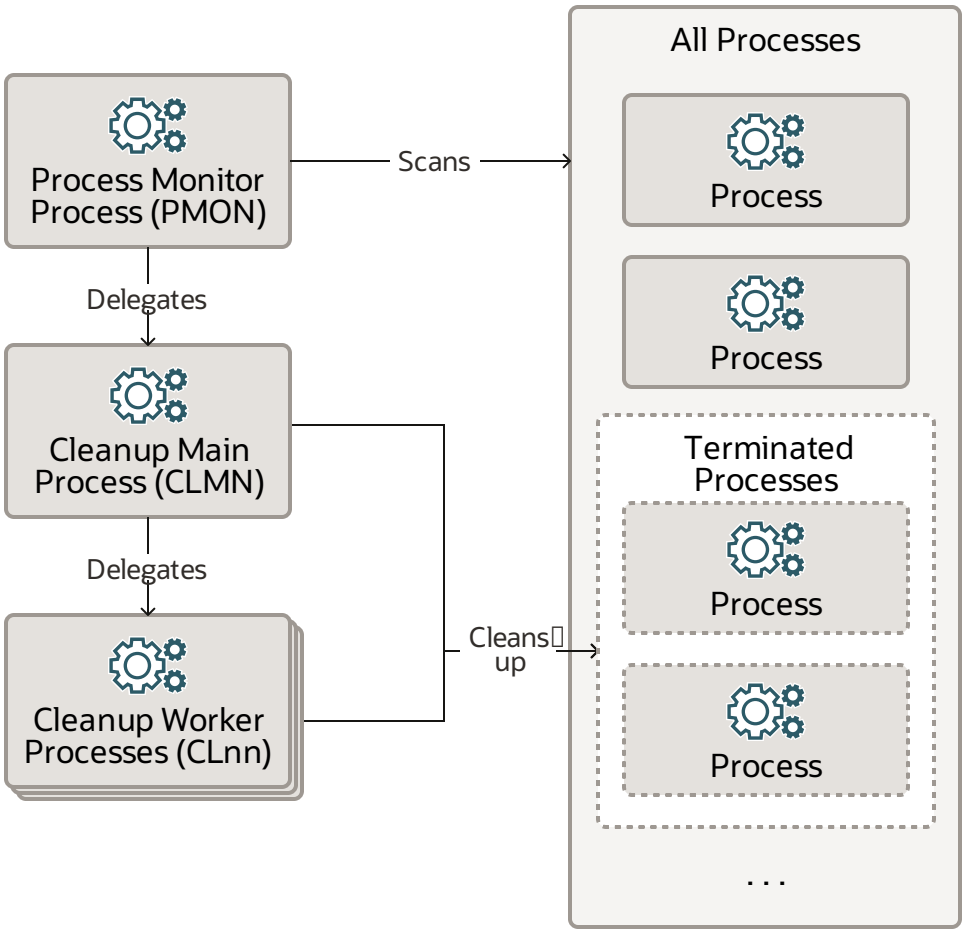
can write backups to sequential devices, such as tape drives. One advantage of backup sets is that RMAN uses unused block compression to save space in backing up data files. The backup set includes only those blocks in the data files that have been used to store data. Backup sets can also be compressed, encrypted, sent to tape. They can use advanced unused-space compression that is not available with data file copies.

RMAN can also 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).

Related Resources

- Backup and Recovery
- Oracle Database Backup Cloud Service
- Zero Data Loss Recovery Appliance

Process Monitor Process (PMON)



Notes

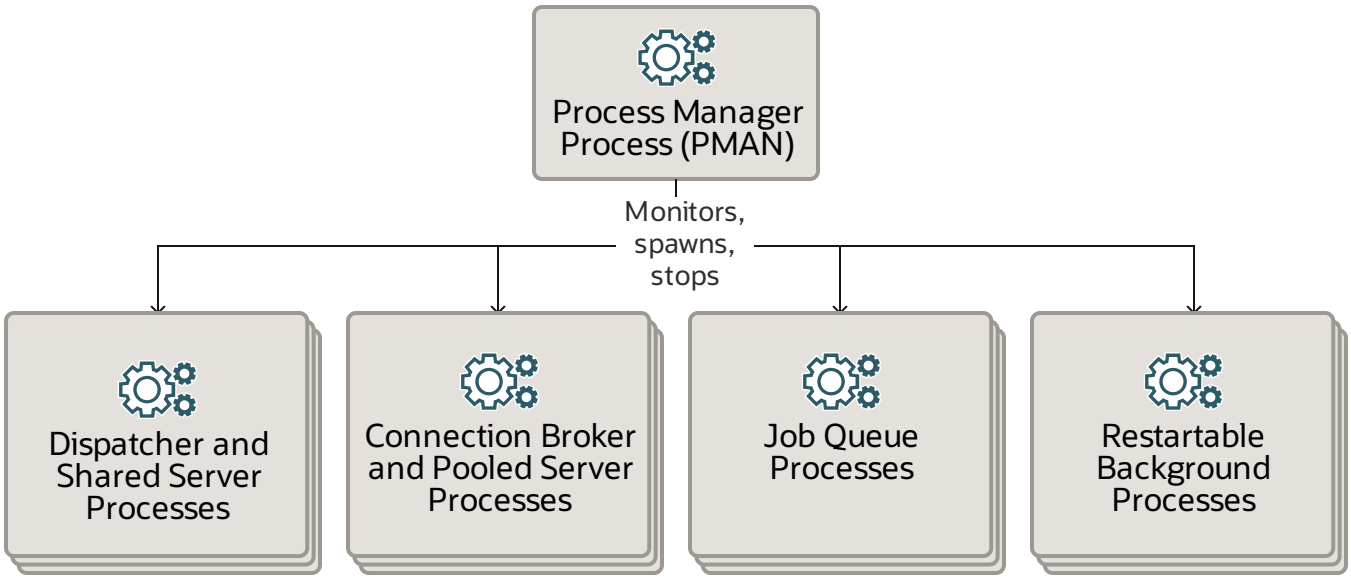
The **process monitor process (PMON)** periodically scans all processes to find any that have terminated abnormally. PMON delegates cleanup work to the **cleanup main process (CLMN)**, which periodically performs cleanup of terminated processes and sessions. CLMN delegates to **cleanup worker processes (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 (Oracle ASM) instances and Oracle ASM Proxy instances.

Related Resources

- Background Processes
- Process Monitor Process (PMON) Group

Process Manager Process (PMAN)



Notes

The **process manager process (PMAN)** oversees several background processes including shared servers, pooled servers, and job queue processes. PMAN monitors, spawns, and stops the following types of processes as needed:

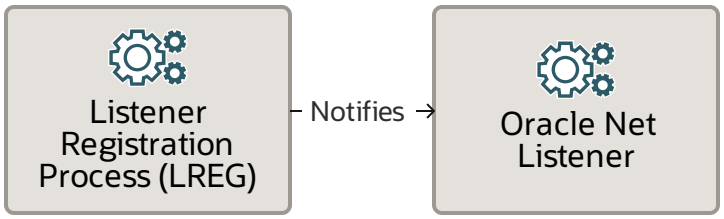
- Dispatcher and shared server processes
- Connection broker and pooled server processes for database resident connection pools
- Job queue processes
- Restartable background processes

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 (Oracle ASM) instances and Oracle ASM proxy instances.

Related Resources

- Background Processes
- Process Manager (PMAN)

Listener Registration Process (LREG)



Notes

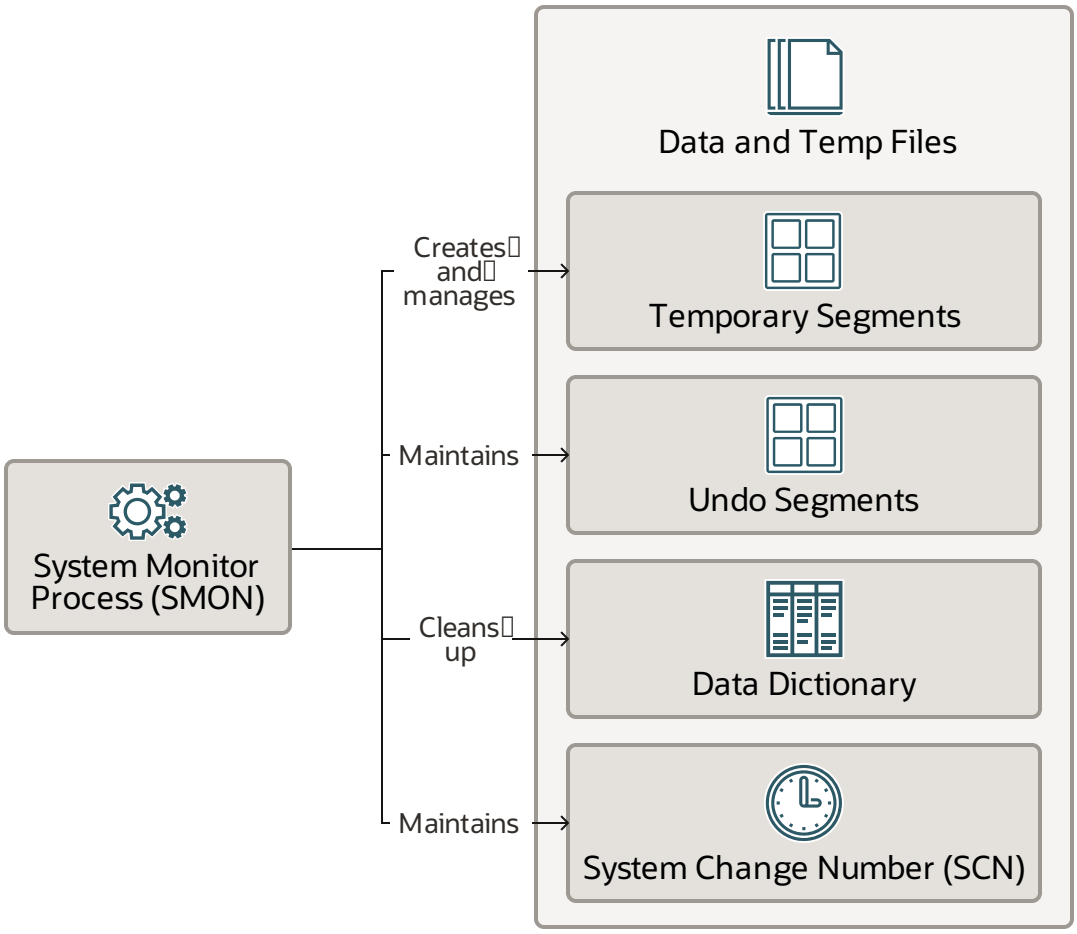
The **listener registration process (LREG)** notifies the listeners about instances, services, handlers, and endpoints.

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

Related Resources

- Background Processes
- Listener Registration Process (LREG)

System Monitor Process (SMON)



Notes

The **system monitor process (SMON)** performs many database maintenance tasks, including the following:

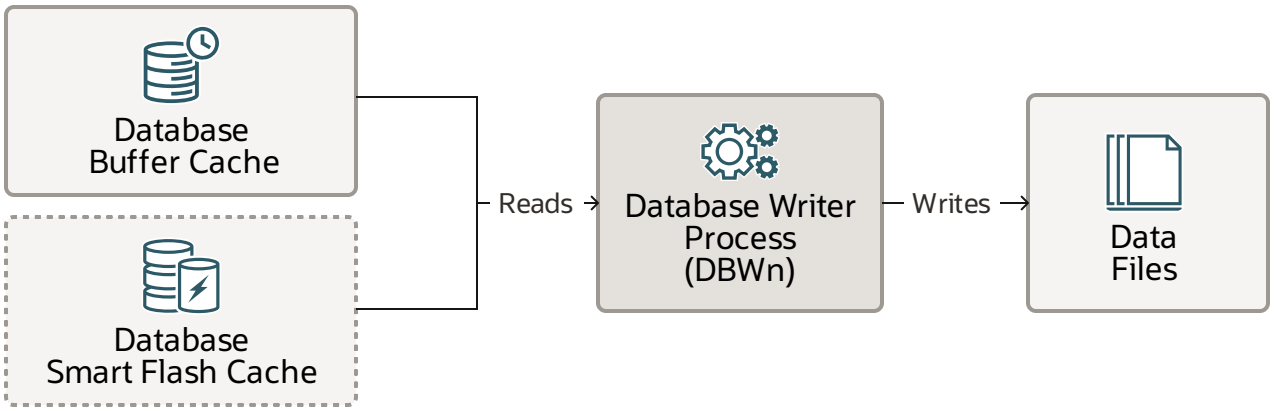
- Creates and manages the temporary tablespace metadata and reclaims space from orphaned temporary segments.
- Maintains the undo tablespace by onlining, offlining, and shrinking the undo segments based on undo space usage statistics.
- Cleans up the data dictionary when it is in a transient and inconsistent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

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

Related Resources

- Background Processes
- System Monitor Process (SMON)

Database Writer Process (DBWn)



Notes

The **database writer process (DBWn)** reads the database buffer cache and writes modified buffers to data files. It also handles checkpoints, file open synchronization, and logging of Block Written records. DBWn also reads the Database Smart Flash Cache (Flash Cache) when Flash Cache is configured.

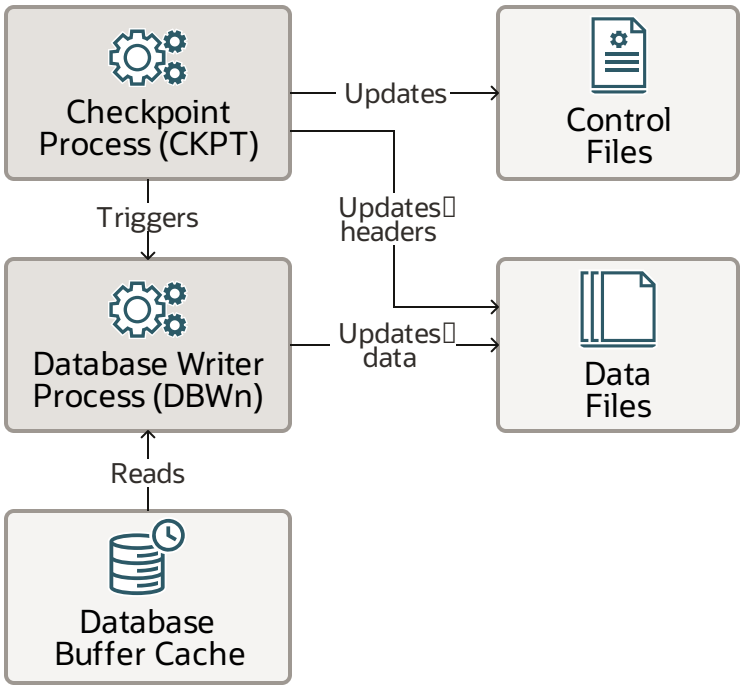
In many cases the blocks that DBWn writes are scattered throughout the disk, so the writes tend to be slower than the sequential writes performed by the log writer process (LGWR). DBWn performs multiblock writes when possible to improve efficiency. The number of blocks that are written in a multiblock 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.

Related Resources

- Background Processes
- Database Writer Process (DBW)

Checkpoint Process (CKPT)



Notes

The **checkpoint process (CKPT)**, at specific times, starts a checkpoint request by triggering the **database writer process (DBWn)** to read the database buffer cache and write modified buffers to data files. On completion of individual checkpoint requests, CKPT updates data file headers and control files to record the most recent checkpoint.

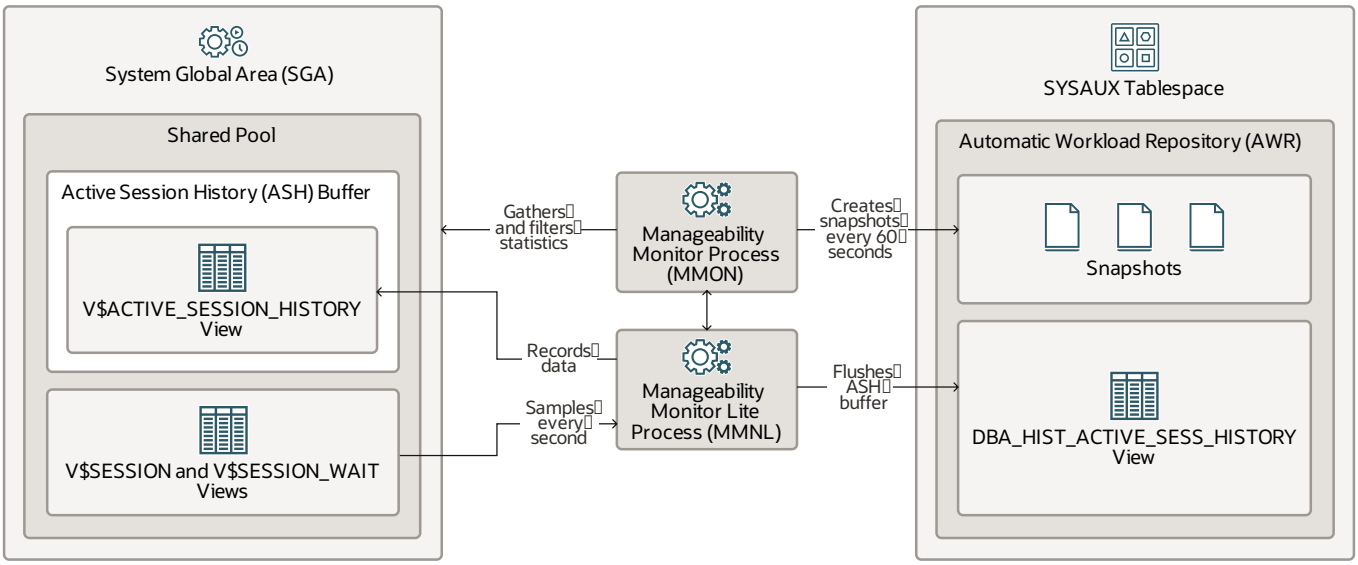
CKPT checks every three seconds to see whether the amount of memory exceeds the value of the PGA_AGGREGATE_LIMIT initialization parameter, and if so, takes action.

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

Related Resources

- Background Processes
- Checkpoint Process (CKPT)

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



Notes

The **manageability monitor process (MMON)** and **manageability monitor lite process (MMNL)** 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 provides problem detection and self-tuning. The AWR resides in the SYSAUX tablespace. AWR reports can be generated in the CDB root or in any PDB. For more information about how the AWR works in a multitenant container database (CDB), see About Using Manageability Features in a CDB.

MMON gathers memory statistics from the **system global area (SGA)**, filters them, and creates snapshots of those statistics in the AWR every 60 minutes (or another interval that you choose). 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 that it's processing) and stores them in the active session history (ASH) buffer, which is part of the shared

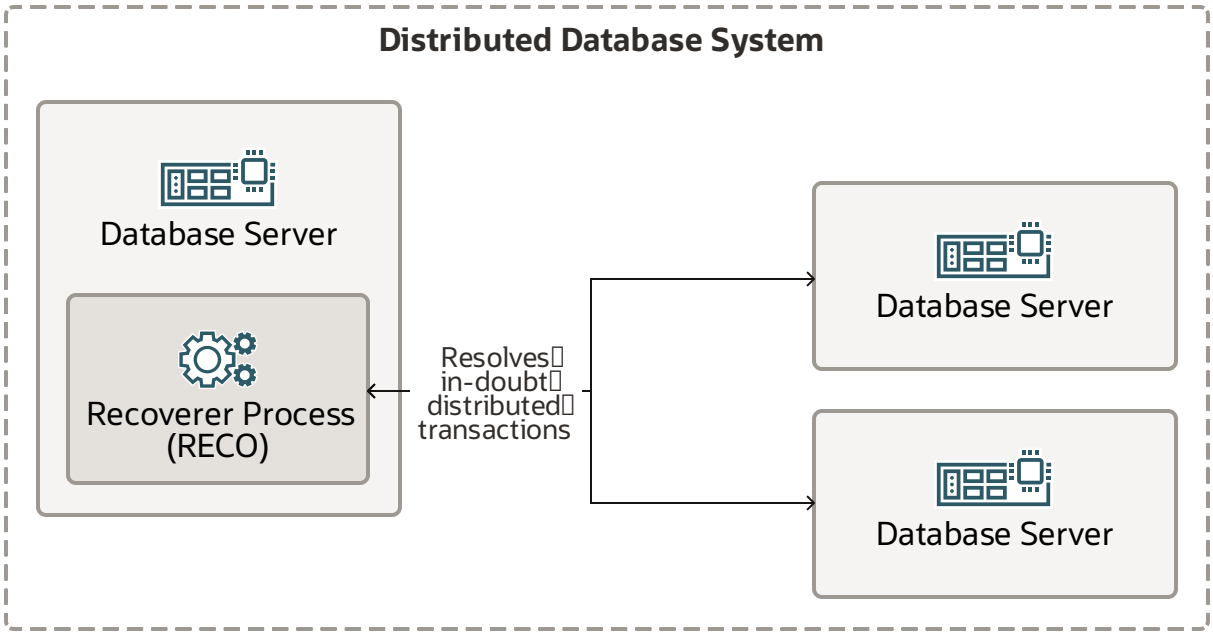
pool in the SGA. Specifically, MMNL samples the V$SESSION and V$SESSION_WAIT views every second and then records that data in the V$ACTIVE_SESSION_HISTORY view in the ASH buffer. MMNL doesn't sample Inactive sessions. The ASH is a rolling buffer in memory, so newer information overwrites earlier information 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, MMNL flushes only one in every 10 entries. 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 Oracle Automatic Storage Management (Oracle ASM) instances.

Related Resources

- Active Session History (ASH)
- Automatic Workload Repository (AWR)
- Background Processes
- Manageability Monitor Processes (MMON and MMNL)

Recoverer Process (RECO)



Notes

The **recoverer process (RECO)** resolves distributed transactions that are pending because of a network or system failure in a distributed database system, which is set of database servers that can appear to applications as a single data source.

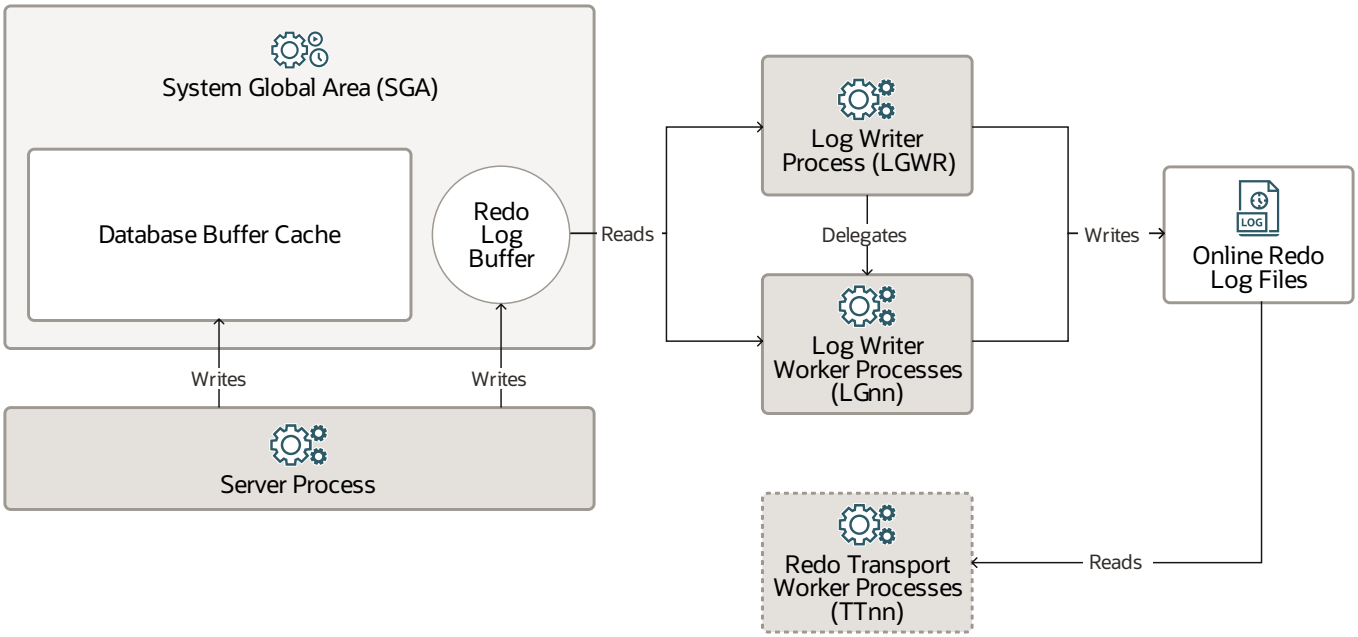
The RECO process of a node automatically connects to other databases involved in an in-doubt distributed transaction. When RECO reestablishes a connection between the databases, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved transactions.

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

Related Resources

- Background Processes
- Recoverer Process (RECO)

Log Writer Process (LGWR)



Notes

Server processes write changes to data blocks in the database buffer cache and write redo data into the redo log buffer. The **log writer process (LGWR)** writes redo log entries sequentially from the redo log buffer to the online redo log. If the database has a multiplexed redo log, then LGWR writes the same redo log entries to all members of a redo log file group.

LGWR handles the operations that are very fast or must be coordinated. It delegates operations that could benefit from concurrent operations to the **log writer worker processes (LGnn)**, which are numbered LG00-LG99. These operations include writing the redo from the log buffer to the redo log file and posting the completed write to the server process that is waiting.

The **redo transport worker processes (TTnn)**, numbered TT00-TTzz, ship redo from the current online and standby redo logs to remote standby destinations that are configured for asynchronous (ASYNC) redo transport.

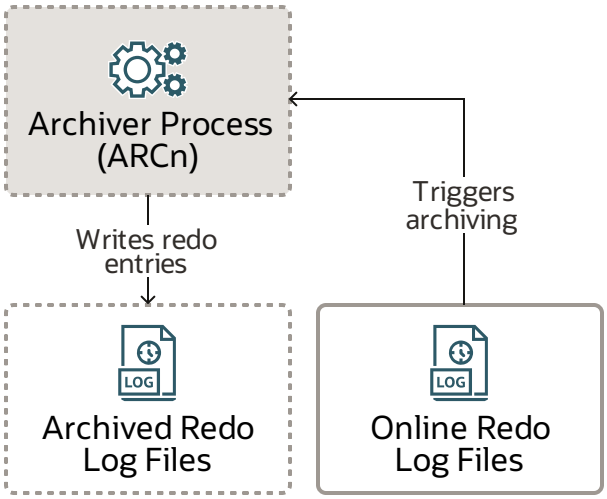
LGWR can run as a thread or as an operating system process. In addition to database instances, LGWR also runs on Oracle Automatic Storage Management (Oracle ASM) instances. Each database instance in an Oracle Real Application Clusters (Oracle RAC) configuration has its own set of redo

log files.

Related Resources

- Background Processes
- Log Writer Process (LGWR)

Archiver Process (ARCn)



Notes

The **archiver processes (ARCn)** exist only when the database is in ARCHIVELOG mode and automatic archiving is enabled, in which case ARCn automatically archives online redo log files. The log writer process (LGWR) cannot reuse and overwrite an online redo log group until it has been archived.

The database starts multiple ARCn 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).

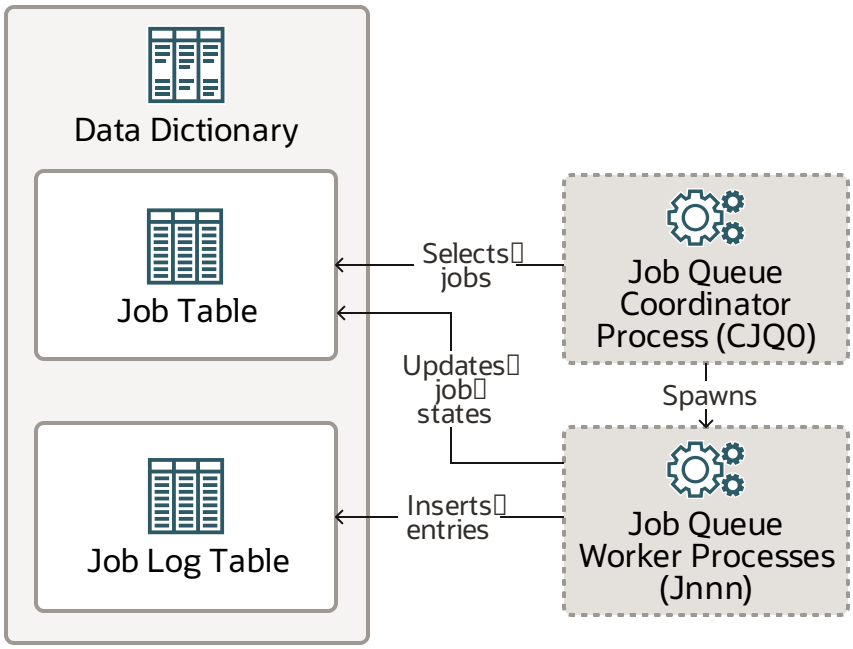
The LOG_ARCHIVE_MAX_PROCESSES initialization parameter specifies the number of ARCn processes that the database initially invokes. If you anticipate a heavy workload for archiving, such as during bulk loading of data, you can increase the maximum number of ARCn processes. There can also be multiple archive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

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

Related Resources

- Archiver Processes (ARCn)
- Background Processes

Job Queue Coordinator Process (CJQ0)



Notes

The **job queue coordinator process (CJQ0)** selects jobs that need to be run from the data dictionary and spawns **job queue worker processes (Jnnn)** to run the jobs. Oracle Scheduler automatically starts and stops CJQ0 as needed. The JOB_QUEUE_PROCESSES initialization parameter specifies the maximum number of processes that can be created for running jobs. CJQ0 starts only as many job queue processes as are required by the number of jobs to run and the available resources.

Jnnn processes run jobs that the job coordinator assigns. When workers pick jobs for processing, they do the following:

- Gather all the metadata that is needed to run the job (for example, program arguments and privilege information).
- Start a database session as the owner of the job, start a transaction, and then start running the job.
- Commit and end the transaction after the job is complete.
- Close 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.

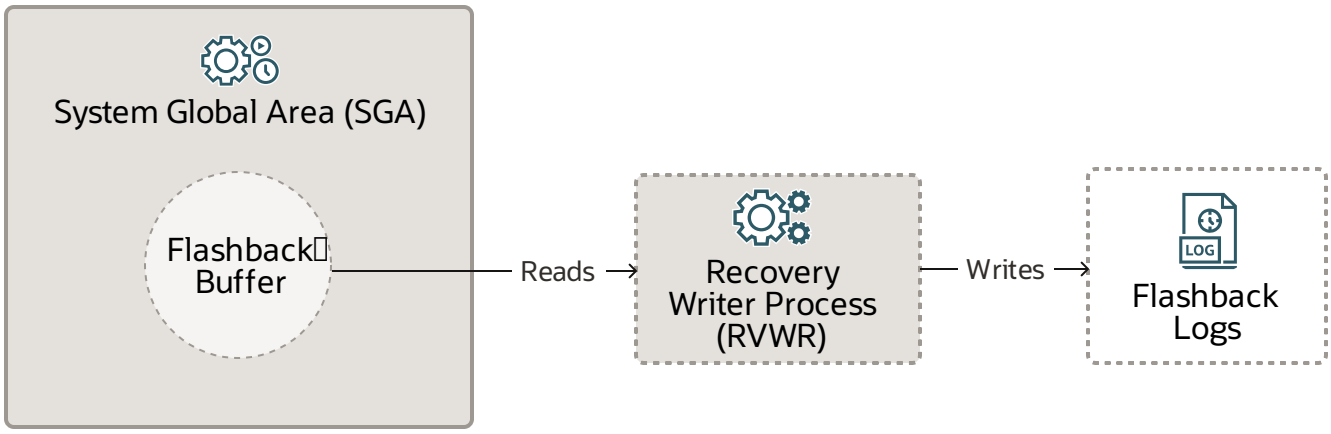
- Insert an entry into the job log table.
- Update the run count and, if necessary, the failure and retry counts.
- Clean up.
- Look for new work. (If there is none, they go to sleep.)

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

Related Resources

- Background Processes
- Job Queue Processes (CJQ0 and Jnnn)

Recovery Writer Process (RVWR)



Notes

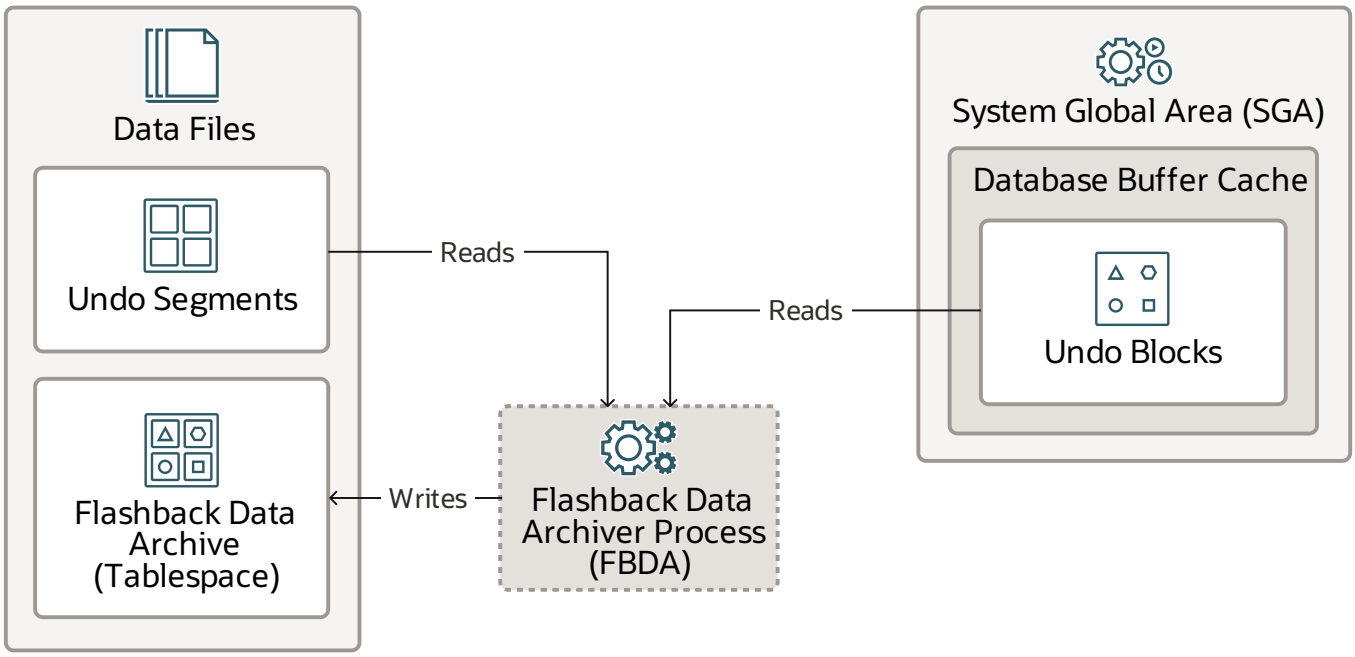
When you use Flashback Database, the **recovery writer process (RVWR)** reads flashback data from the flashback buffer in the system global area (SGA) and writes to the flashback logs. That is, it undoes transactions from the current state of the database to a time in the past, provided that you have the required flashback logs.

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

Related Resources

- Background Processes

Flashback Data Archiver Process (FBDA)



Notes

When you use Flashback Database, the **flashback data archiver process (FBDA)** tracks and stores transactional changes to a table over its lifetime. This way, you can flashback tables to restore them to the way they were at a time in the past.

When a transaction that modifies a tracked table commits, FBDA reads undo blocks in the database buffer cache and undo segments in data files. Then it filters what is relevant to objects that are marked for archival and copies that undo information into the flashback data archive (tablespace) in the data files. FBDA maintains metadata on the current rows and tracks how much data has been archived.

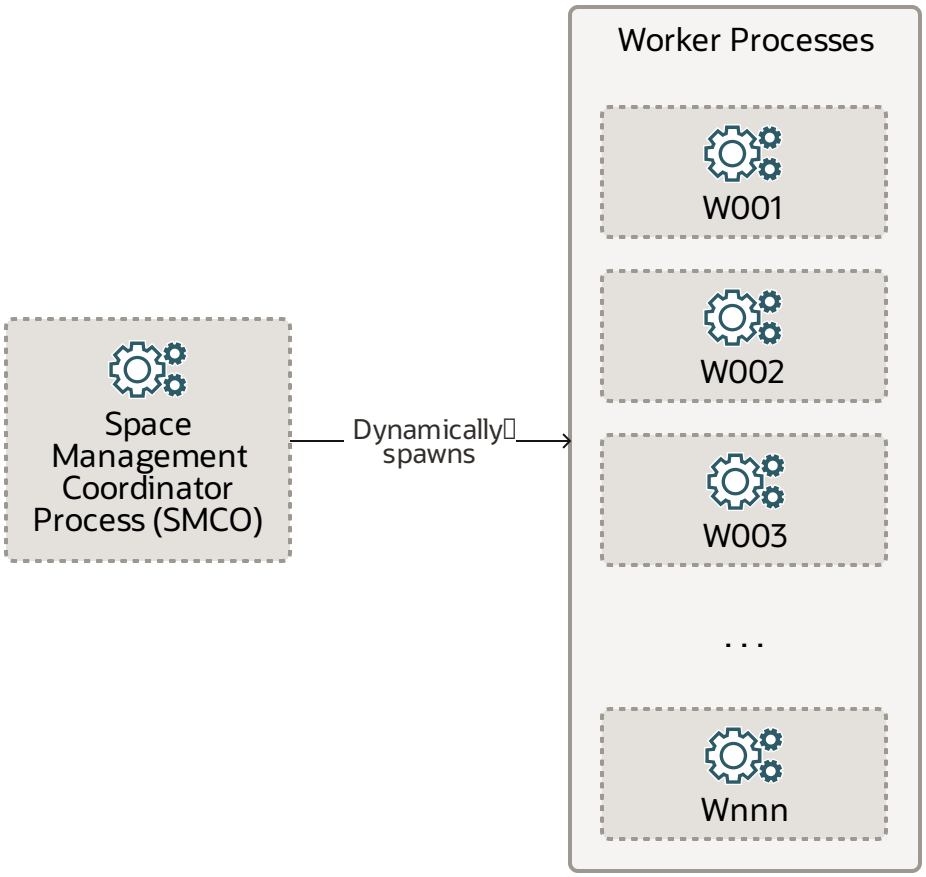
FBDA automatically manages the flashback data archive for space, organization (partitioning tablespaces), and retention. Additionally, FBDA keeps track of how far the archiving of tracked transactions has progressed.

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

Related Resources

- Background Processes
- Flashback Data Archive Process (FBDA)

Space Management Coordinator Process (SMCO)



Notes

The optional **space management coordinator process (SMCO)** schedules various space management tasks, including proactive space allocation and space reclamation. SMCO dynamically spawns **space management worker processes (Wnnn)** to implement these tasks. After starting, the worker acts as an autonomous agent. After it finishes a task, the worker process automatically picks up another task from the queue. The process terminates itself after being idle for a long time.

Wnnn worker processes (named W001, W002, and so on) perform tasks on behalf of Space Management and the Oracle Database In-Memory option.

For Space Management, Wnnn processes perform space management tasks in the background, including the following:

- Preallocate space into locally managed tablespace and SecureFiles segments based on space usage growth analysis.
- Reclaim space from dropped segments.
- Perform fast ingest deferred inserts.

For the Oracle Database In-Memory option, Wnnn processes perform the following tasks:

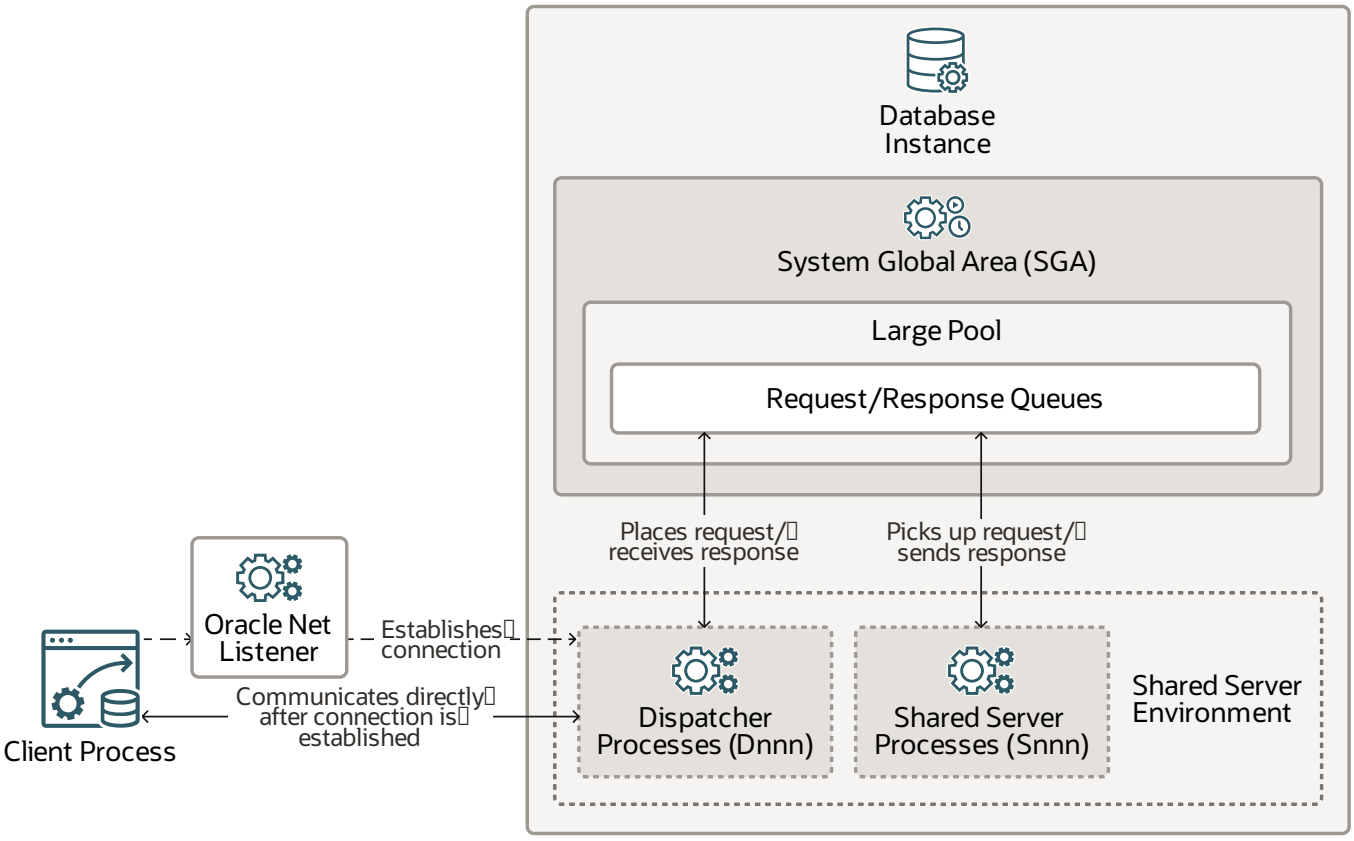
- Prepopulate in-memory enabled objects with priority LOW/MEDIUM/HIGH/CRITICAL and repopulate in-memory objects for the in-memory coordinator background process (IMCO).
- Initiate in-memory populate and repopulate tasks for IMCO foreground processes in response to queries and DMLs that reference in-memory enabled objects.

Both SMCO and Wnnn can run as threads or as operating system processes.

Related Resources

- Background Processes
- Space Management Coordinator Process (SMCO)

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



Notes

In a shared server environment, a **dispatcher process (Dnnn)** directs multiple incoming network session requests to a pool of **shared server processes (Snnn)**. You can create multiple dispatcher processes for a single database instance.

The **Oracle Net Listener** process establishes a connection to the dispatcher. When a client process makes a connection request that requires a shared server process, the listener returns the dispatcher address so the client process can communicate with the dispatcher directly after the connection is established.

The dispatcher places the client request in the **request queue** in the large pool of the system global area (SGA) within the database instance. The next available shared server process picks up the request and processes the request. When the request is complete, the shared server process places the response on the calling dispatcher's **response queue** in the large pool, and the response queue sends the response to the dispatcher. The dispatcher returns the completed request to the client

process.

Both Snnn and Dnnn can run as threads or as operating system processes. In addition to database instances, Dnnn also runs in a shared server environment.

Related Resources

- Background Processes
- Dispatcher Processes (Dnnn)
- Shared Server Architecture
- Shared Server Processes (Snnn)