ORACLE
University

Oracle Database 23ai: Administration Workshop

Student Guide

S1106066GC10

Learn more from Oracle University at ***education.oracle.com***

1012202023

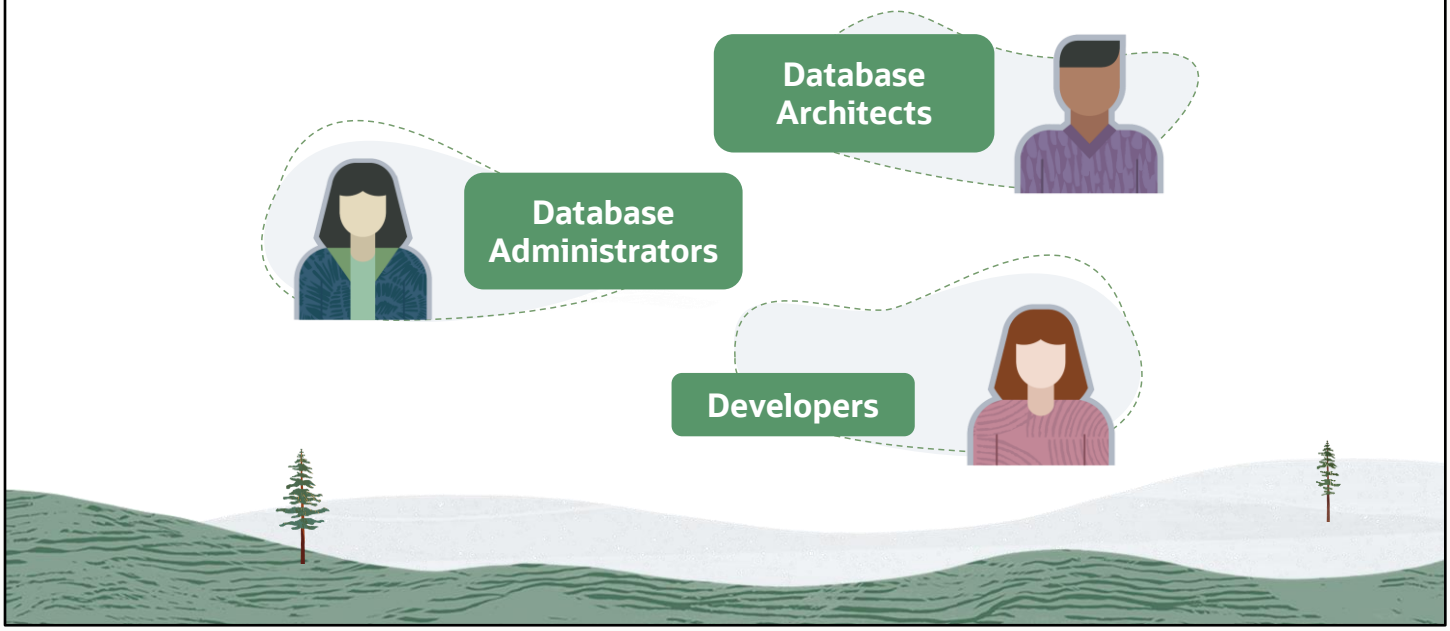
Contents

ORACLE
University

Oracle Database 23c: Administration Workshop

Course Overview

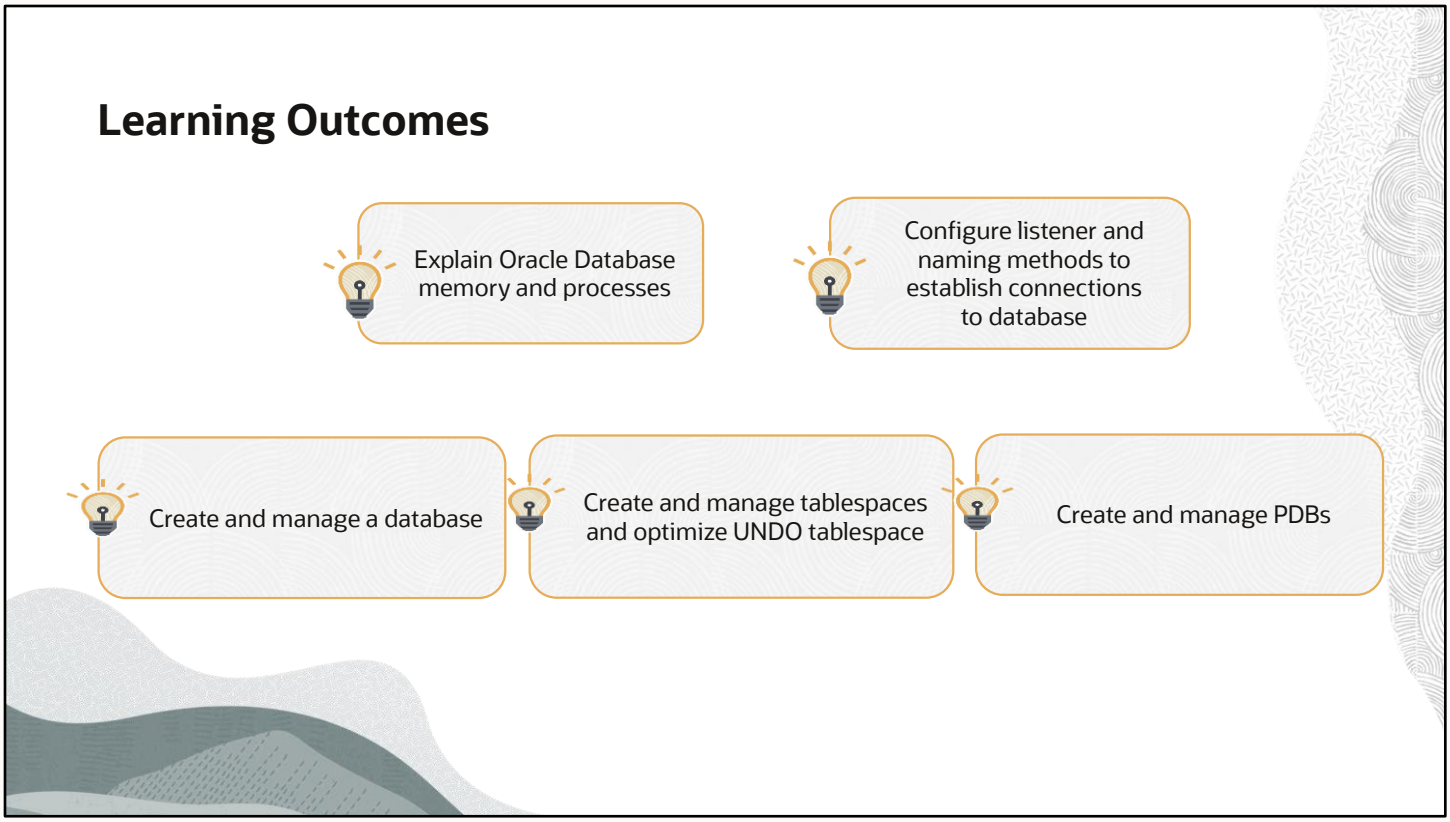
Target Audience



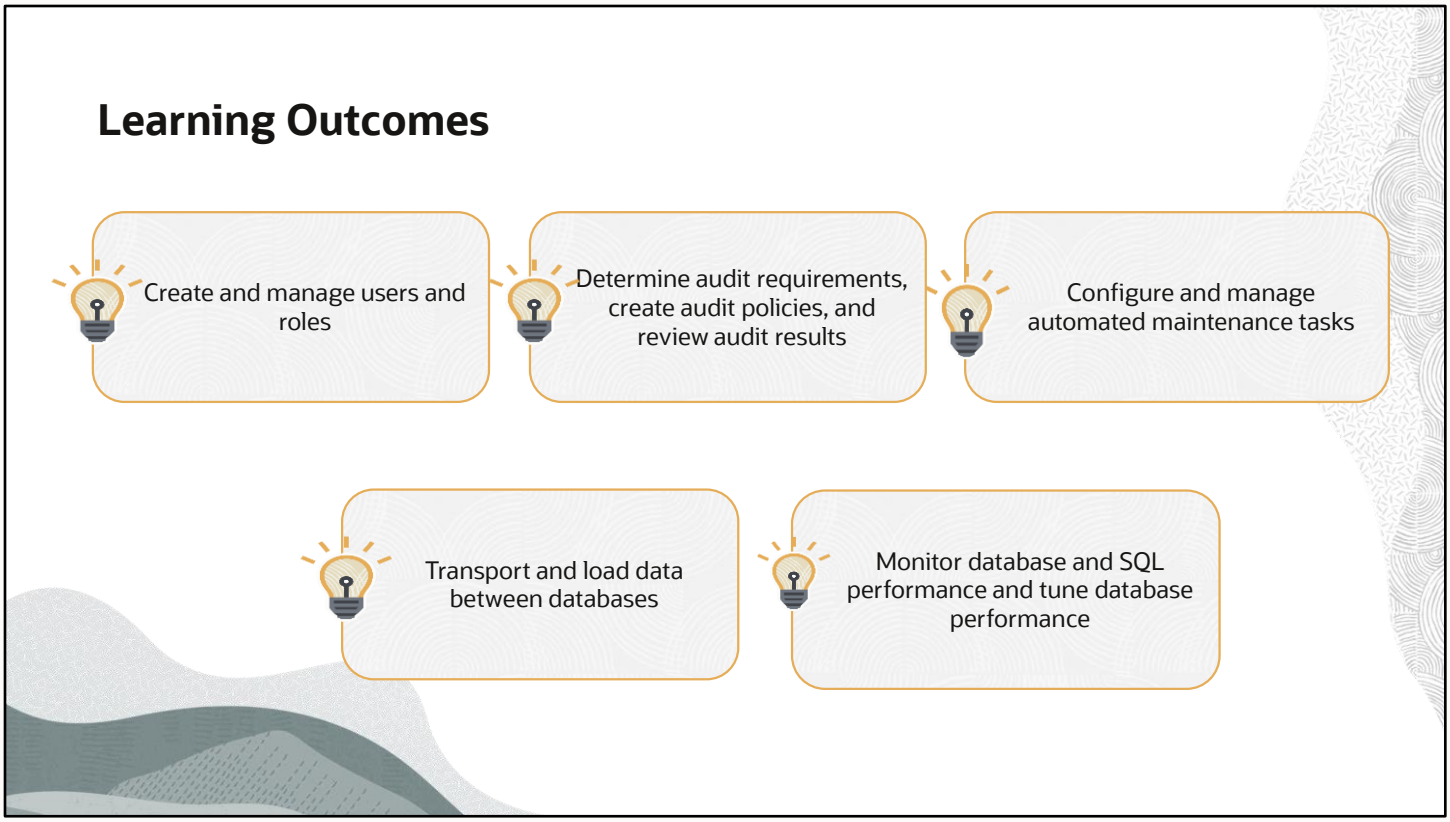
Prerequisites

- Familiarity with computers
- Basic understanding of databases

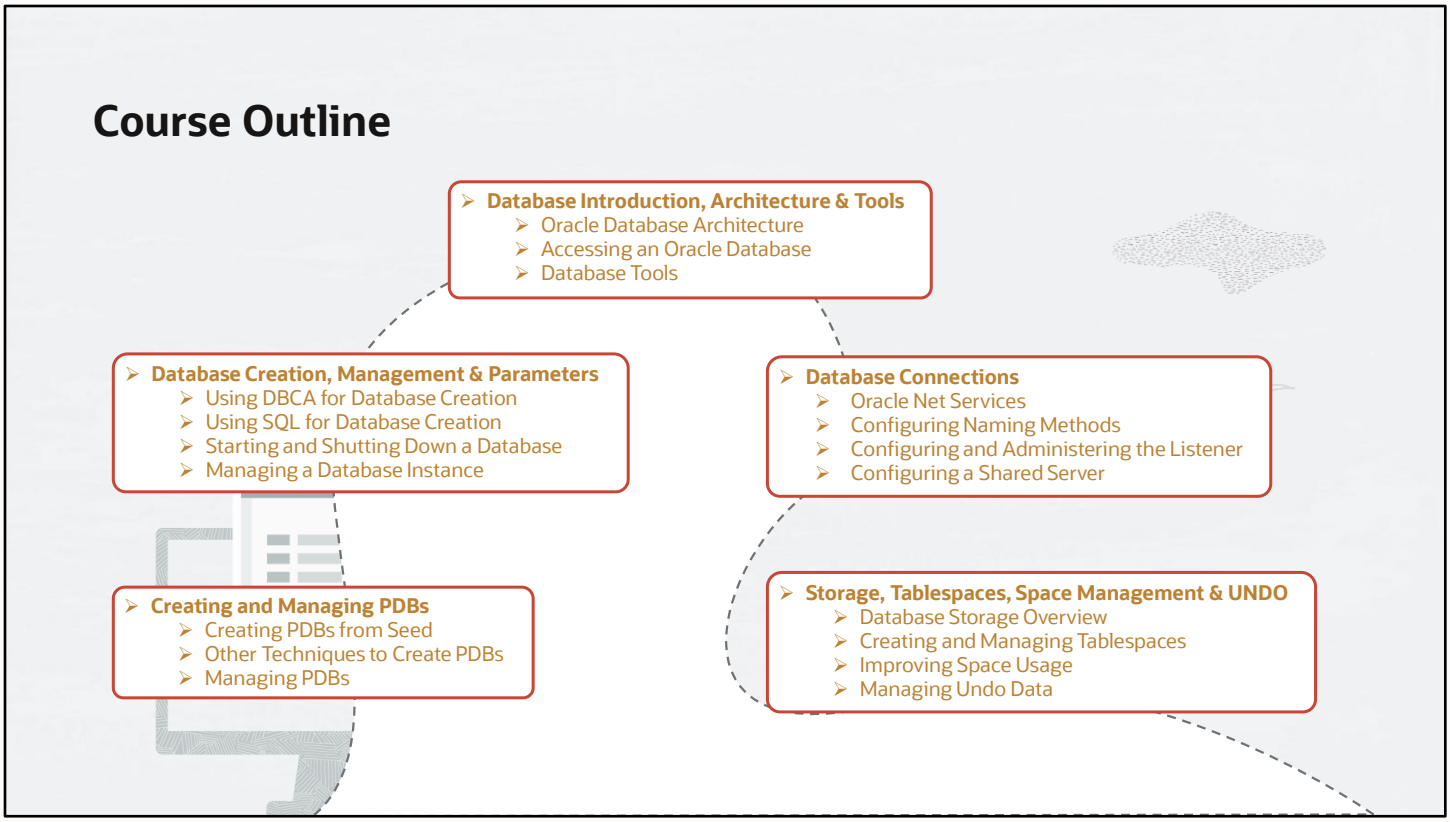
Learning Outcomes



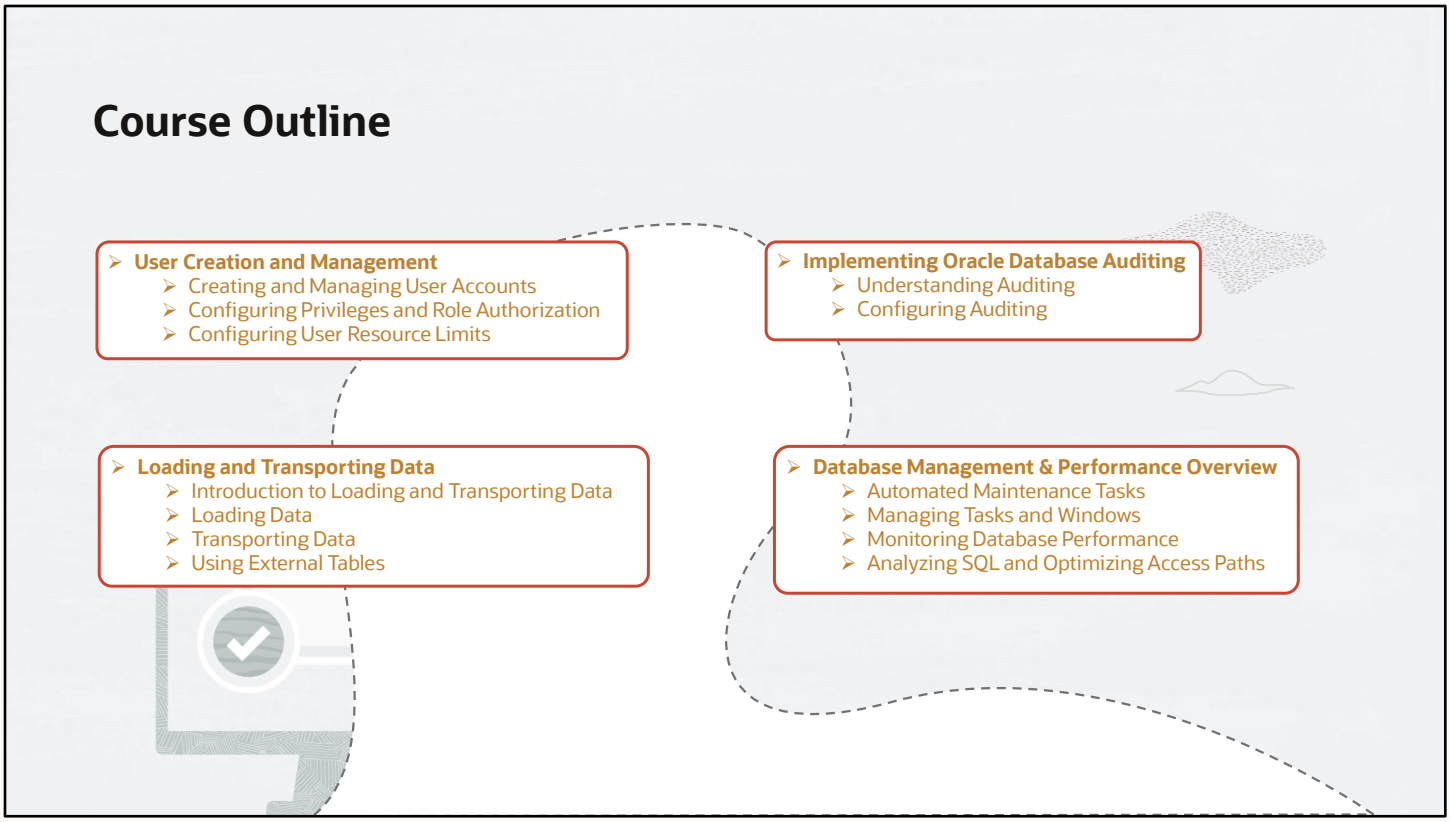
Learning Outcomes



Course Outline

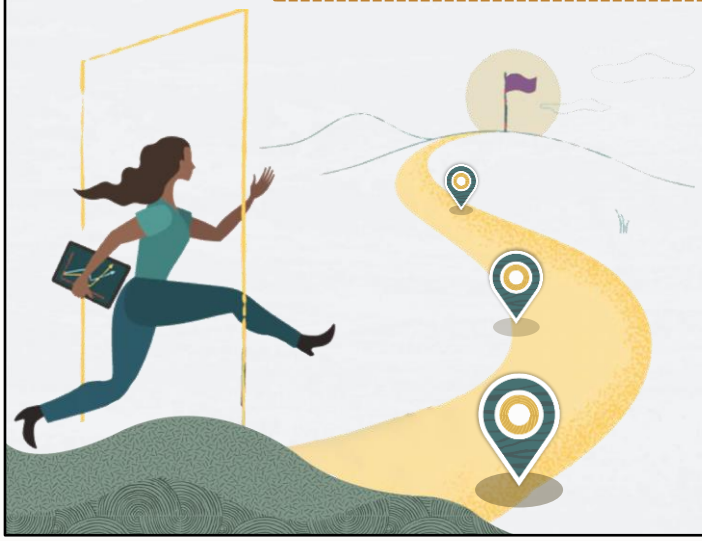


Course Outline

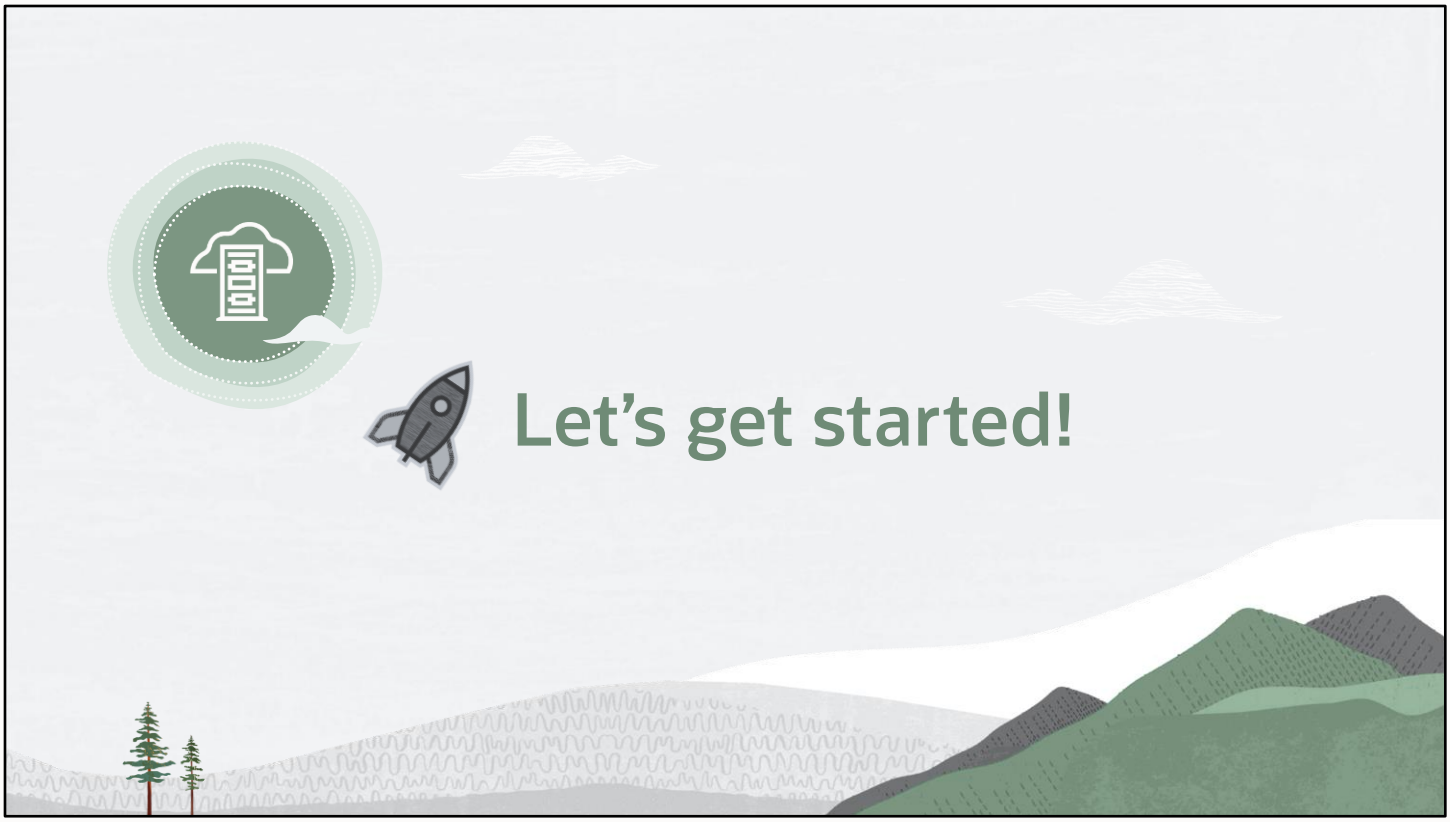


What's Next?

Oracle Database 23c: Backup and Recovery



Oracle Database Administration 2023 Certified Professional Credential


Let’s get started!

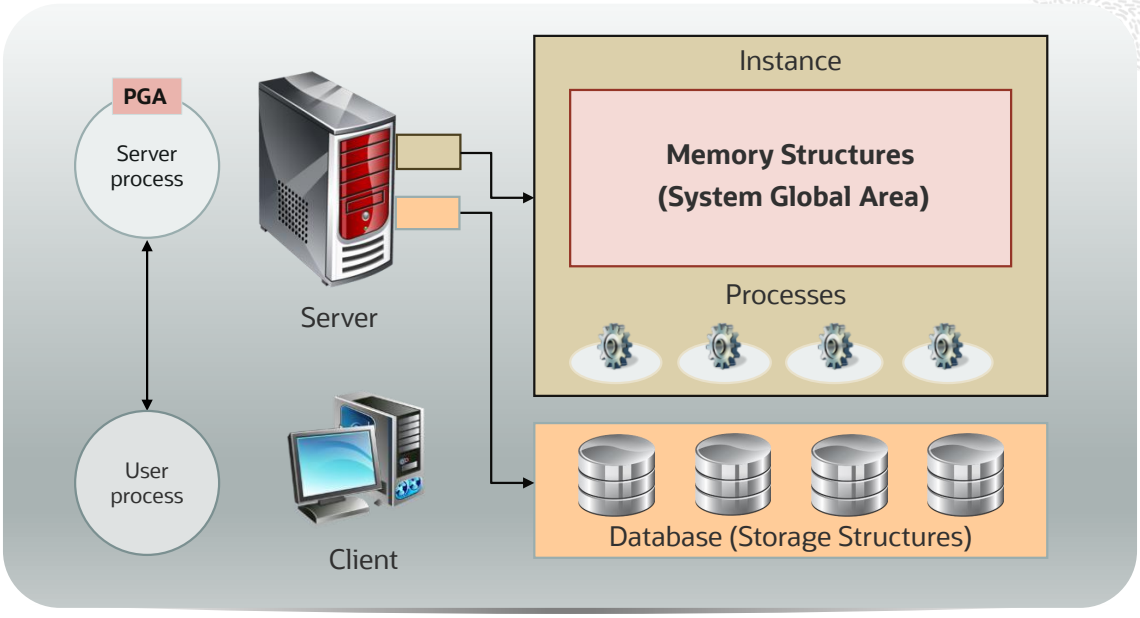
ORACLE
University

Introduction to Oracle Database

Objectives

- List the major architectural components of Oracle Database
- Describe multitenant architecture
- Explain memory structures
- Define process architecture
- Describe database sharding

Oracle Database Server Architecture: Overview

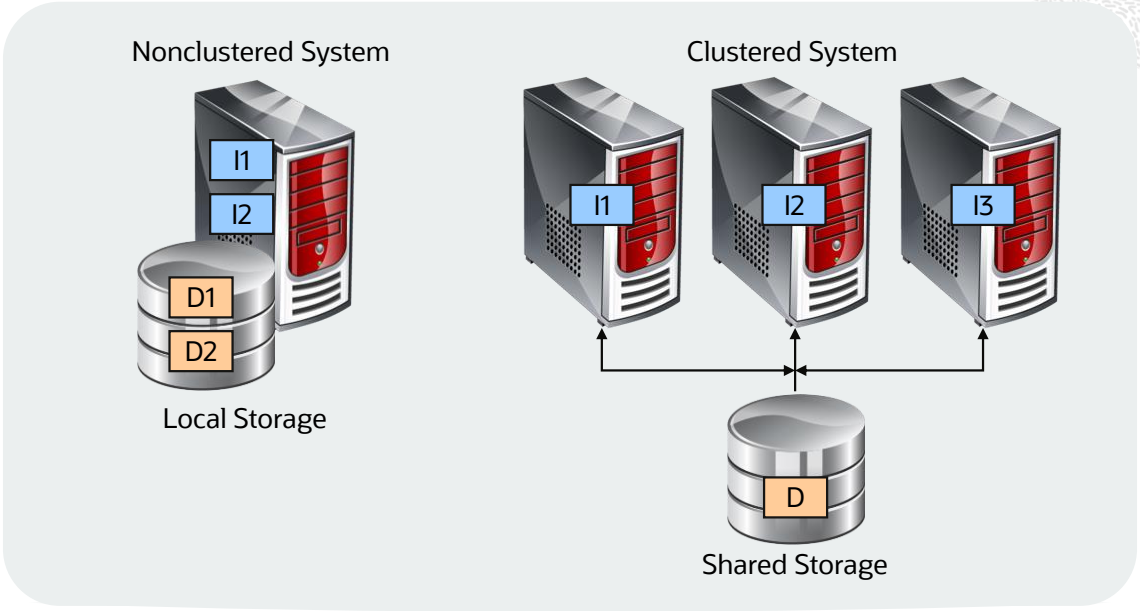


There are three major structures in Oracle Database server architecture: memory structures, processes, and storage structures. A basic Oracle Database system consists of an Oracle database and a database instance.

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

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

Oracle Database Instance Configurations



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

Oracle Multitenant Container Database: Introduction

- **Container**: A logical collection of data or metadata within the multitenant architecture
- **Pluggable database (PDB)**: A portable collection of schemas, schema objects, and nonschema objects
- Containers in a **multitenant container database (CDB)**:
  - CDB root container (also known as the **root**)
  - System container (includes the root and all PDBs)
  - Application containers
  - Root PDB
  - User-created PDBs

At the physical level, the multitenant container database (CDB) has a database instance and database files, just as a noncontainer database does.

A CDB avoids redundancy of:

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

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

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

Oracle Multitenant Container Database: Introduction

- All pluggable databases share:
  - Background processes
  - Shared memory management and some memory structures
  - Some of the Oracle metadata

At the physical level, the multitenant container database (CDB) has a database instance and database files, just as a noncontainer database does.

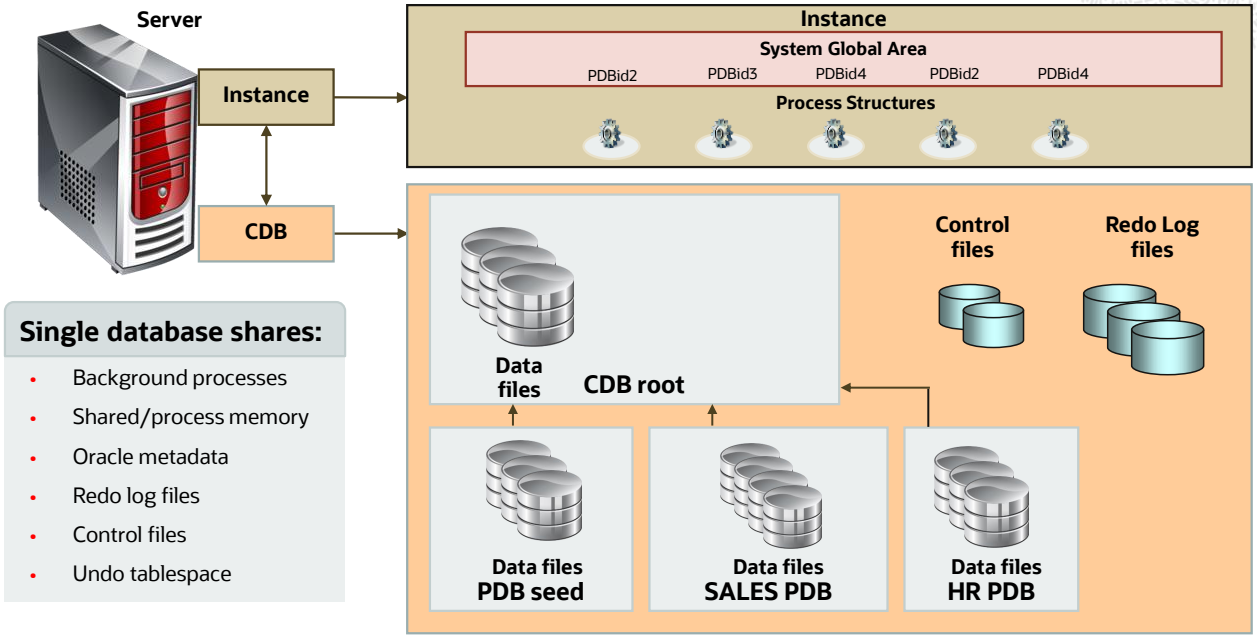
A CDB avoids redundancy of:

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

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

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

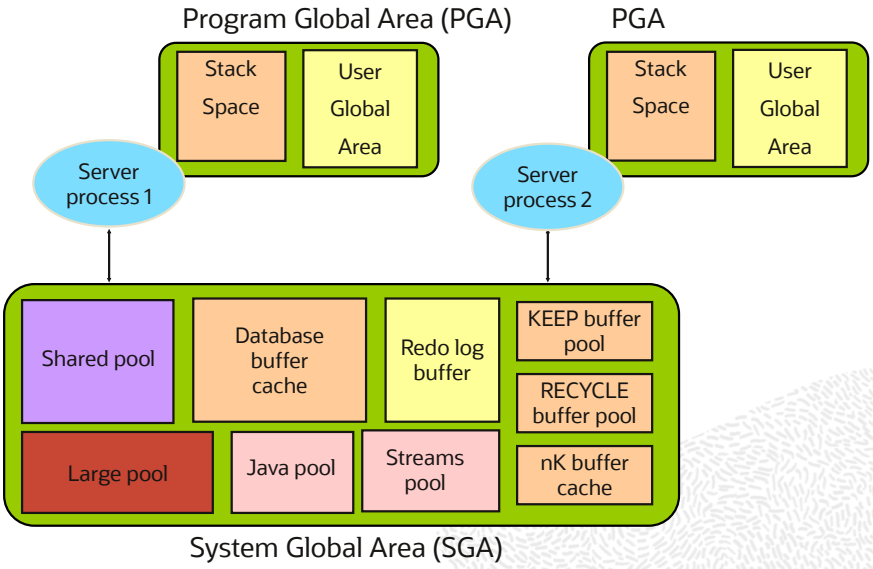
Oracle Multitenant Container Database: Architecture



You can design your database to be a multitenant container database (CDB). A CDB, as illustrated in the slide, is made up of one root container, one seed pluggable database (seed PDB), and one or more user-created pluggable databases (simply referred to as PDBs). To a user or application, PDBs appear logically as separate databases.

- The root container, named `CDB$ROOT`, contains multiple data files. The data files store Oracle-supplied metadata and common users (users that are known in every container). This information is shared with all PDBs.
- The seed PDB, named `PDB$SEED`, is a system-supplied PDB template containing multiple data files that you can use to create new PDBs.
- The user-created 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. For example, in the slide, there are two PDBs—one for the sales organization (named SALES) and another for the Human Resources department (named HR). 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.

Oracle Database Memory Structures



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

Two basic memory structures are associated with an instance:

- **System Global Area (SGA):** Group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- **Program Global Areas (PGA):** Memory regions that contain data and control information for a server or background process. A PGA is nonshared memory created by Oracle Database when a server or background process is started. Access to the PGA is exclusive to the server process. Each server process and background process has its own PGA.

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

- **Shared pool:** Caches various constructs that can be shared among users
- **Database buffer cache:** Caches blocks of data retrieved from the database
- **KEEP buffer pool:** A specialized type of database buffer cache that is tuned to retain blocks of data in memory for long periods of time
- **RECYCLE buffer pool:** A specialized type of database buffer cache that is tuned to recycle or remove block from memory quickly
- **nK buffer cache:** One of several specialized database buffer caches designed to hold block sizes different than the default database block size
- **Redo log buffer:** Caches redo information (used for instance recovery) until it can be written to the physical redo log files stored on the disk
- **Large pool:** Optional area that provides large memory allocations for certain large processes, such as Oracle backup and recovery operations, and I/O server processes
- **Java pool:** Used for all session-specific Java code and data in the Java Virtual Machine (JVM)
- **Streams pool:** Used by Oracle Streams to store information required by capture and apply

When you start the instance by using Enterprise Manager or SQL*Plus, the amount of memory allocated for the SGA is displayed.

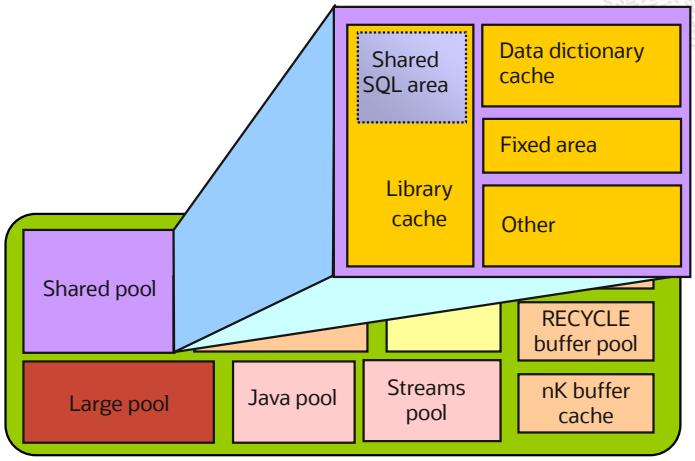
A Program Global Area (PGA) is a memory region that contains data and control information for each server process. An Oracle server process services a client's requests. Each server process has its own private PGA that is allocated when the server process is started. Access to the PGA is exclusive to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

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

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

Shared Pool

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



System Global Area (SGA)

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

The **data dictionary** is a collection of database tables and views containing reference information about the database, its structures, and its users. Oracle Database accesses the data dictionary frequently during SQL statement parsing. This access is essential to the continuing operation of Oracle Database.

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

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

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

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

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

Individual SQL statements contained in a PL/SQL program unit are processed just like other SQL statements. Despite their origins in a PL/SQL program unit, these SQL statements use a shared area to hold their parsed representations and a private area for each session that runs the statement.

The SQL query result cache and PL/SQL function result cache are new to Oracle Database 23c. They share the same infrastructure, appear in the same dynamic performance (`v$`) views, and are administered using the same supplied package.

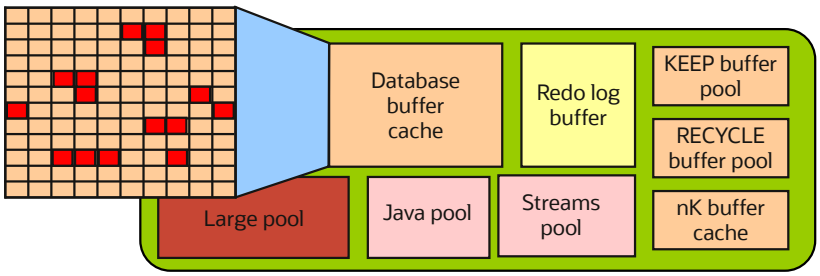
Results of queries and query fragments can be cached in memory in the SQL query result cache. The database can then use cached results to answer future executions of these queries and query fragments. Because retrieving results from the SQL query result cache is faster than rerunning a query, frequently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

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

The fixed area of the shared pool represents startup overhead for the SGA. It is very small in comparison to a typically sized shared pool or SGA.

Database Buffer Cache

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



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

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

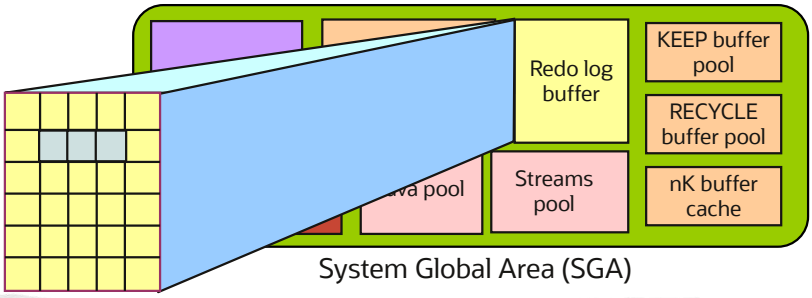
The buffers in the cache are managed by a complex algorithm that uses a combination of least recently used (LRU) lists and touch count. The LRU helps to ensure that the most recently used blocks tend to stay in memory to minimize disk access.

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

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

Redo Log Buffer

- Is a circular buffer in the SGA
- Holds information about changes made to the database
- Contains redo information
  - Changes made by DML and DDL

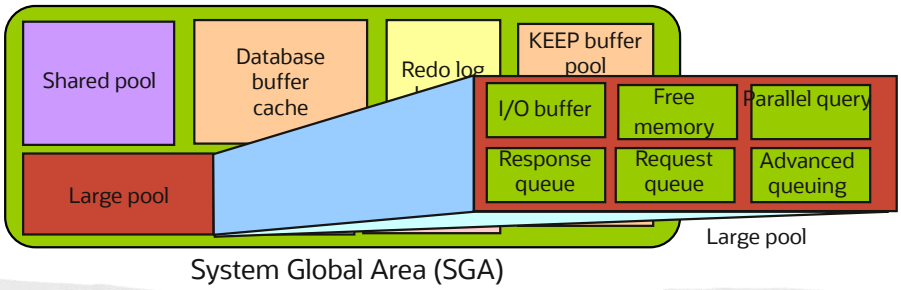


The redo log buffer is a circular buffer in the SGA that holds information about changes made to the database. This information is stored in redo entries. Redo entries contain the information necessary to reconstruct (or redo) changes that are made to the database by DML, DDL, or internal operations.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

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

Large Pool

- Provides large memory allocations for:
  - Session memory for the shared server
  - I/O server processes
  - Oracle Database backup and restore operations



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

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

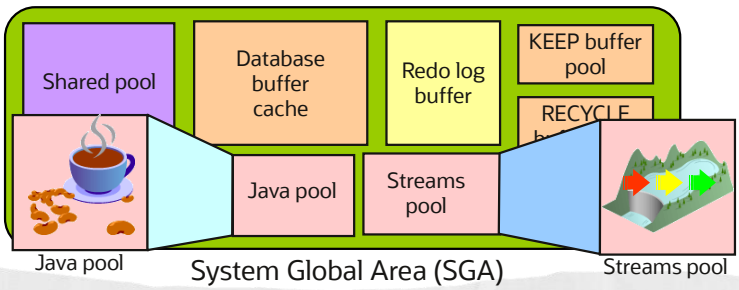
By allocating session memory from the large pool for shared server, Oracle XA, or parallel query buffers, Oracle Database can use the shared pool primarily for caching shared SQL and avoid the performance overhead that is caused by shrinking the shared SQL cache.

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

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

Java Pool and Streams Pool

- Java pool memory is used to store all session-specific Java code and data in the JVM.
- Streams pool memory is used exclusively by Oracle Streams to:
  - Store buffered queue messages
  - Provide memory for Oracle Streams processes



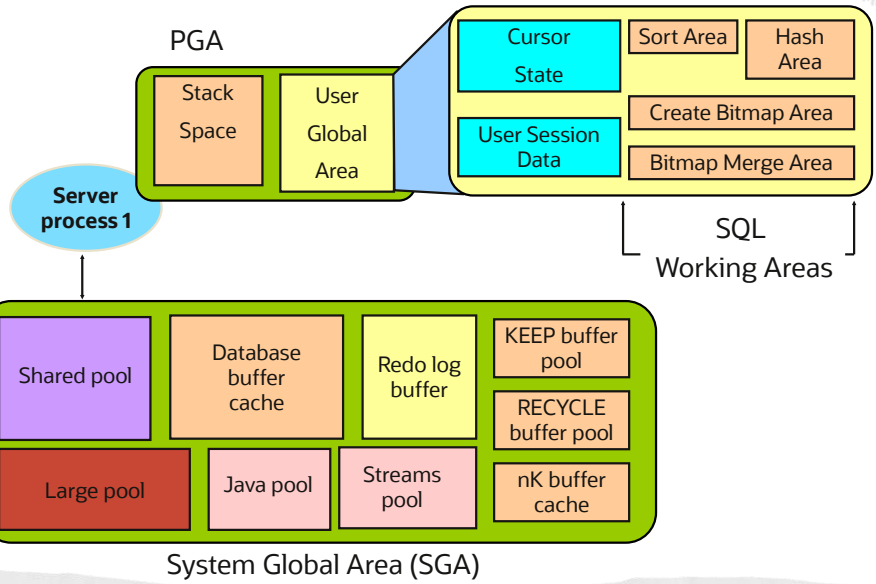
Java pool memory is used to store all session-specific Java code and data in the JVM. Java pool memory is used in different ways, depending on the mode in which Oracle Database is running.

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

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

Note: A detailed discussion of Java programming and Oracle Streams is beyond the scope of this class.

Program Global Area (PGA)



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

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

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

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

Process Architecture

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

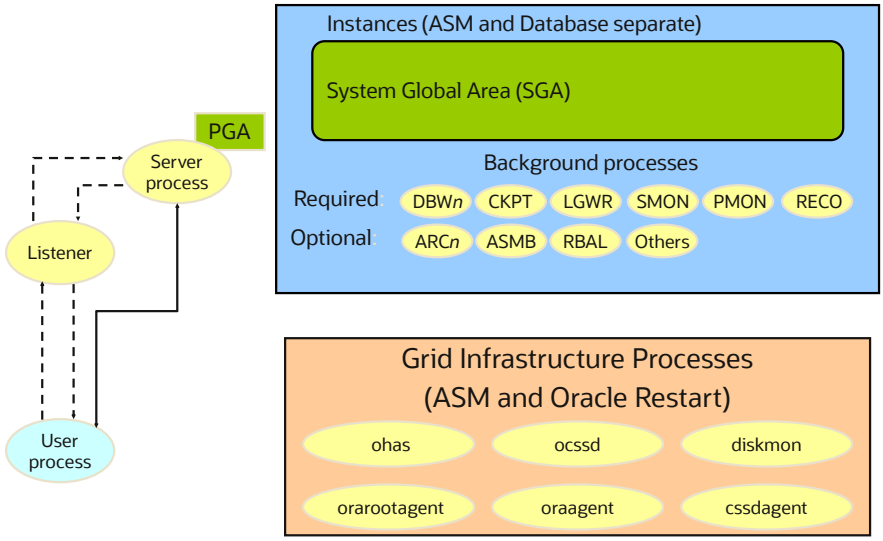
The processes in an Oracle database system can be divided into three major groups:

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

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

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

Process Structures



Server Processes

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

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

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

Background Processes

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

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

- Database writer process (DBW*n*)
- Log writer process (LGWR)
- Checkpoint process (CKPT)
- System monitor process (SMON)
- Process monitor process (PMON)
- Recoverer process (RECO)
- Job queue coordinator (CJQ0)
- Job slave processes (J*nnn*)
- Archiver processes (ARC*n*)
- Queue monitor processes (QMN*n*)

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

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

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

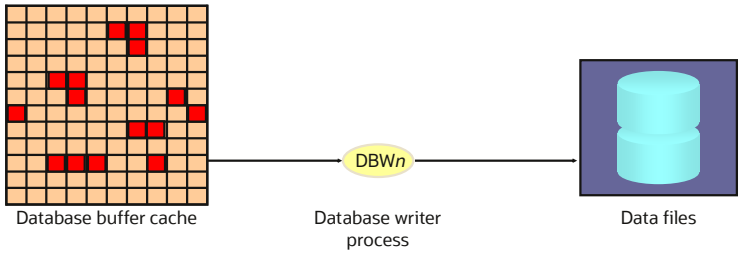
Oracle Grid Infrastructure processes on Linux and UNIX systems include the following:

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

Note: For a more detailed list of the background processes, please consult the *Oracle Background Processes* appendix in this course or the *Oracle Database Reference* guide.

Database Writer Process (DBWn)

- Writes modified (dirty) buffers in the database buffer cache to disk:
  - Asynchronously while performing other processing
  - To advance the checkpoint



The Database Writer process (DBW*n*) writes the contents of buffers to data files. The DBW*n* processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk. Although one Database Writer process (DBW*0*) is adequate for most systems, you can configure additional processes (DBW*1* through DBW*9* and DBW*a* through DBW*z*) to improve write performance if your system modifies data heavily. These additional DBW*n* processes are not useful on uniprocessor systems.

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

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

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

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

The `DB_WRITER_PROCESSES` initialization parameter specifies the number of DBW*n* processes. The maximum number of DBW*n* processes is 36. If it is not specified by the user during startup, Oracle Database determines how to set `DB_WRITER_PROCESSES` based on the number of CPUs and processor groups.

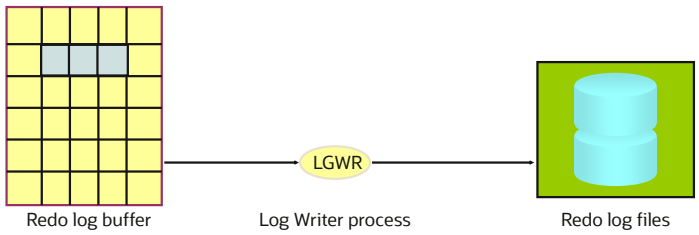
The DBW*n* process writes dirty buffers to disk under the following conditions:

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

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

Log Writer Process (LGWR)

- Writes the redo log buffer to a redo log file on disk
- Writes:
  - When a user process commits a transaction
  - When the redo log buffer is one-third full
  - Before a DBWn process writes modified buffers to disk
  - Every three seconds



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

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

LGWR writes:

- When a user process commits a transaction
- When the redo log buffer is one-third full
- Before a DBW*n* process writes modified buffers to disk (if necessary)
- Every three seconds

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

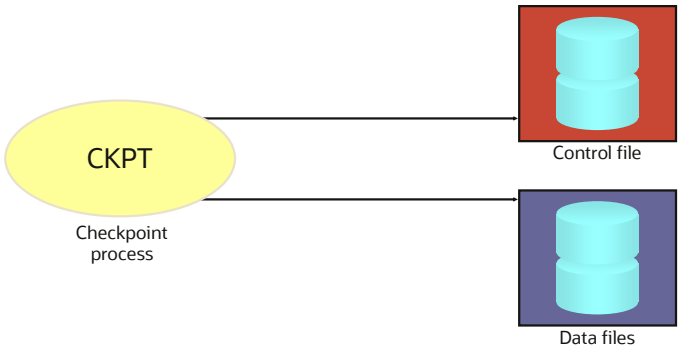
When a user issues a `COMMIT` statement, LGWR puts a commit record in the redo log buffer and writes it to disk immediately, along with the transaction's redo entries. The corresponding changes to data blocks are deferred until it is more efficient to write them. This is called a **fast commit mechanism**.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

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

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

Checkpoint Process (CKPT)

- Records checkpoint information in
  - Control file
  - Each data file header

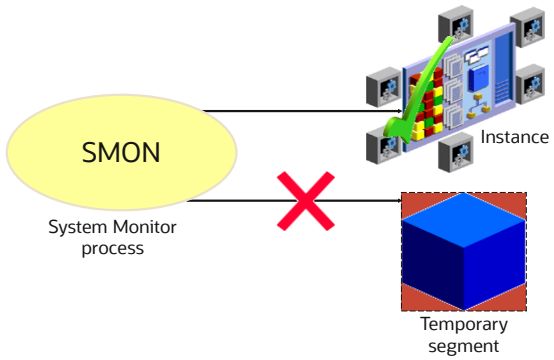


A **checkpoint** is a data structure that defines a system change number (SCN) in the redo thread of a database. Checkpoints are recorded in the control file and in each data file header. They are a crucial element of recovery.

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

System Monitor Process (SMON)

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

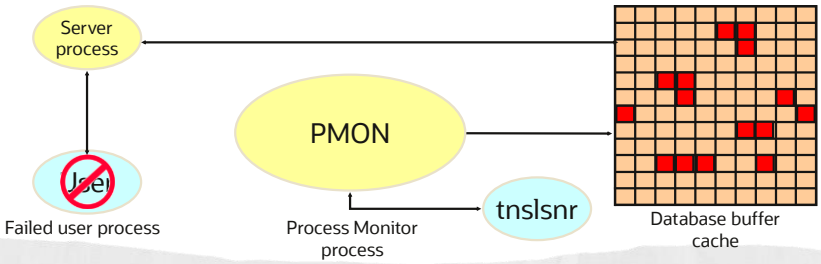


The System Monitor process (SMON) performs recovery at instance startup if necessary. SMON is also responsible for cleaning up temporary segments that are no longer in use. If any terminated transactions were skipped during instance recovery because of file-read or offline errors, SMON recovers them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

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

Process Monitor Process (PMON)

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



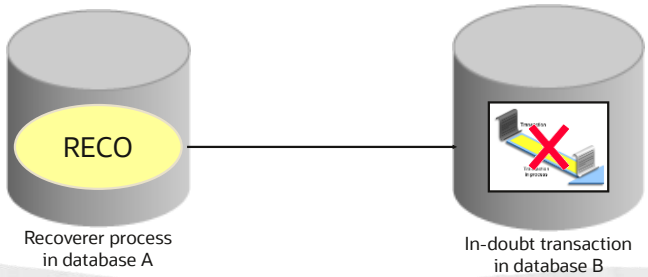
The Process Monitor process (PMON) performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process was using. For example, it resets the status of the active transaction table, releases locks, and removes the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

PMON periodically checks the status of dispatcher and server processes, and restarts any that have stopped running (but not any that Oracle Database has terminated intentionally). PMON also registers information about the instance and dispatcher processes with the network listener.

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

Recoverer Process

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



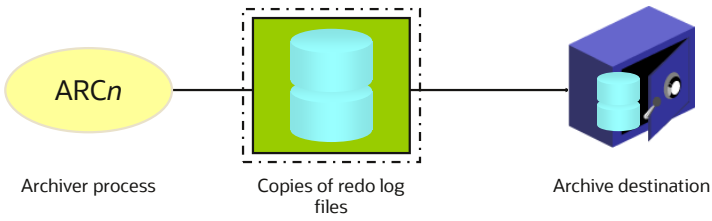
Recoverer Process (RECO)

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

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

Archiver Processes (ARCn)

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

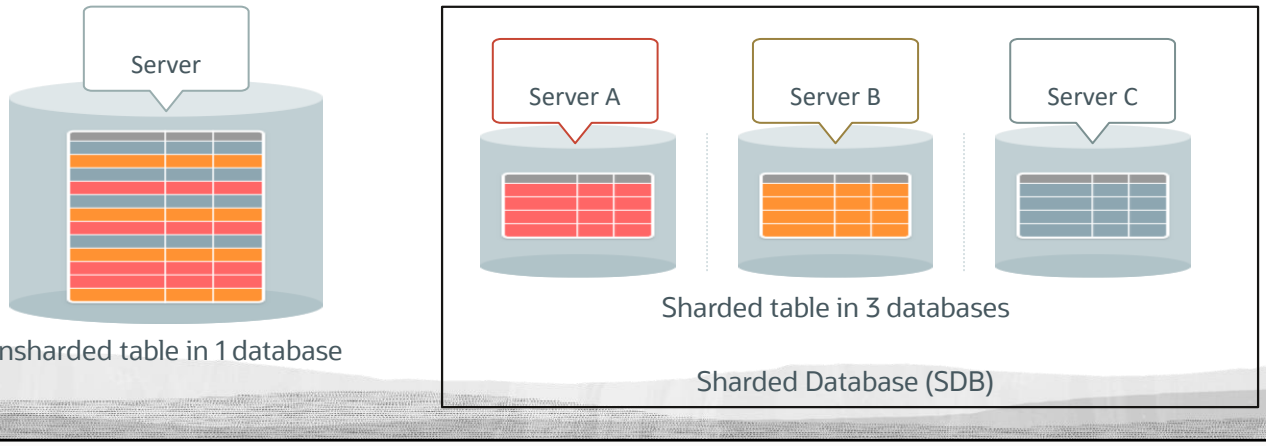


The archiver processes (ARC*n*) copy redo log files to a designated storage device after a log switch has occurred. ARC*n* processes are present only when the database is in `ARCHIVELOG` mode and automatic archiving is enabled.

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

Database Sharding: Introduction

- A shared-nothing architecture for scalability and availability
- Horizontally partitioned data across independent databases
- Loosely coupled data tier without clusterware



Sharding is a data tier architecture where data is horizontally partitioned across independent databases. Each database in such a configuration is called a **shard**. All shards together make up a single logical database, which is referred to as a **sharded database** or SDB.

Horizontal partitioning involves splitting a database table across shards so that each shard contains the table with the same columns but a different subset of rows. The diagram in the slide shows an unsharded table on the left with the rows represented by different colors. On the right, the same table data is shown horizontally partitioned across three shards or independent databases. Each partition of the logical table resides in a specific shard. We refer to such a table as a **sharded table**.

Sharding is a shared-nothing database architecture because shards do not share physical resources such as CPU, memory, or storage devices. Shards are also loosely coupled in terms of software; they do not run clusterware.

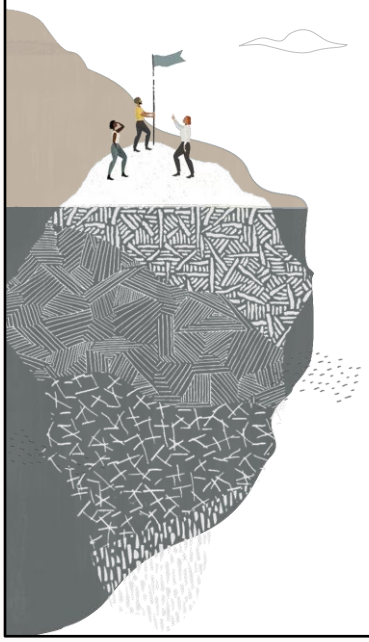
From a database administrator's perspective, an SDB consists of multiple databases that can be managed either collectively or individually. However, from an application developer's perspective, an SDB looks like a single database: the number of shards and the distribution of data across them are completely transparent to database applications.

Oracle Database Server: Interactive Architecture Diagram

Access the Interactive Architecture Diagram on the Oracle Help Center Oracle Database “What’s New” page.

https://tinyurl.com/yyepn9ma

Summary

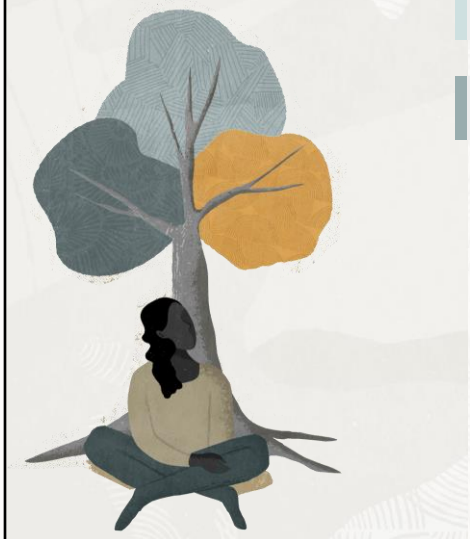


- List the major architectural components of Oracle Database
- Describe multitenant architecture
- Explain memory structures
- Define process architecture
- Describe database sharding

ORACLE
University

Accessing an Oracle Database

Objectives



- Connect to Oracle Database
- Describe the tools used to access Oracle Database

Connecting to an Oracle Database Instance

- You connect client applications to an Oracle Database by connecting to its database instance, not its database.
- A user session is a logical entity that represents the state of the current user login to the database instance.
- Examples of connecting to an Oracle Database instance:
  - By using operating system authentication

```
$ sqlplus / as sysdba
```

  - By using Easy Connect Syntax

```
SQL> connect hr/hr@host1.example.com:1521/db.example.com
```

Connections Compared with Sessions

You connect client applications to database instances (not databases).

A connection is the physical communication pathway between a client process and a database instance.

A user session is a logical entity that represents the state of the current user login to the database instance. A session lasts from the time the user is authenticated by the database instance until the time the user disconnects or exits the client application.

Connecting to CDBs by Using Operating System Authentication

As a database administrator, you can quickly start SQL*Plus and connect to a root container without a password by using the `$ sqlplus / as sysdba` command. This command enables you to connect to the database as the `SYS` user with the `SYSDBA` privilege. There are some rules: You must be on the same machine as the database instance, and the current operating system user must be a member of the privileged `OSDBA` group.

Connecting to PDBs by Using the Easy Connect Syntax

There are many ways to connect to a PDB; however, using the Easy Connect syntax in SQL*Plus is the easiest because it's already enabled on the database server by default and doesn't require any client-side configuration. This syntax supports TCP protocol only (no SSL). It offers no support for advanced connection options such as connect-time failover, source routing, and load balancing.

Easy Connect connection strings take the following form:

```
SQL> CONNECT <username>/<password>@<listener hostname>:<listener port>/<service
name>
```

For example, the `SYSTEM` user requests a connection to the database service named `db.example.com`. The listener is located on a machine named `host01.example.com` and listens on port 1521.

```
SQL> CONNECT hr/hr@host1.example.com:1521/db.example.com
```

If you're starting from a command prompt, you can start SQL*Plus and log in at the same time:

```
$ sqlplus hr/hr@host1.example.com:1521/db.example.com
```

The listener port and service name are optional. If the listener port is not provided, Oracle Net assumes that the default port of 1521 is being used. If the service name is not provided, Oracle Net assumes that the database service name and host name provided in the connect string are identical. For example, assuming that the listener uses TCP to listen on port 1521, the connection string above can be shortened.

For example, a connection string like this:

```
SQL> CONNECT hr/hr@db.example.com:1521/db.example.com
```

...can be shortened to:

```
SQL> CONNECT hr/hr@ db.example.com
```

Disconnecting from the Database Instance

Use the `EXIT` command to exit SQL*Plus, disconnect from the database instance, and end all sessions in the database instance memory.

Oracle Database Tools

- Oracle Database tools each have their own purpose, and some operations can be performed in more than one tool.
- Choosing a tool for a task often comes down to personal preference.
- Tools include:
  - SQL*Plus
  - SQL Developer
  - SQL Developer Command Line (SQLcl)
  - Database Configuration Assistant (DBCA)
  - Oracle Enterprise Manager Database Express (EM Express)

Oracle Database includes the following tools:

- **SQL*Plus:** Use this command-line tool to access a database.
- **SQL Developer:** Use this graphical user interface (GUI) tool to access a database and perform DBA actions.
- **SQL Developer Command Line (SQLcl):** Use this tool to access a database.
- **Database Configuration Assistant (DBCA):** Use this GUI tool to create databases. To use DBCA, you must be on the server to launch the tool from the operating system that houses the CDB.
- **Oracle Enterprise Manager Database Express (EM Express):** Use this GUI tool to perform database administration tasks on one database instance at a time.
- **Oracle Enterprise Manager Cloud Control (EM Cloud Control):** Use this GUI tool to perform database administration tasks on several targets (database instances, listeners, and so on) at the same time.
- **Oracle Management Cloud Database Express (OMX):** Use this GUI to perform database administration tasks. OMX is a replacement for EM Express with a subset of functionality in the initial release.
- **Others:** Many specialized utilities are used to assist with database administration. The ones used in this course may include Listener Control (`lsnrctl`), Oracle Net Configuration Assistant (`netca`), Oracle Net Manager (`netmgr`), ADR Command Interpreter (`adcri`), SQL*Loader (`sqlldr`), Oracle Data Pump Import (`impdp`), and Oracle Data Pump Export (`expdp`). This list does not cover all the utilities available.

Oracle Database Tools

- Oracle Enterprise Manager Cloud Control
- Oracle Management Cloud Database Express (OMX)
- Others such as Listener Control, Oracle Net Configuration Assistant, Oracle Net Manager, ADR Command Interpreter, SQL*Loader, Oracle Data Pump Import, and Oracle Data Pump Export

Oracle Database includes the following tools:

- **SQL*Plus:** Use this command-line tool to access a database.
- **SQL Developer:** Use this graphical user interface (GUI) tool to access a database and perform DBA actions.
- **SQL Developer Command Line (SQLcl):** Use this tool to access a database.
- **Database Configuration Assistant (DBCA):** Use this GUI tool to create databases. To use DBCA, you must be on the server to launch the tool from the operating system that houses the CDB.
- **Oracle Enterprise Manager Database Express (EM Express):** Use this GUI tool to perform database administration tasks on one database instance at a time.
- **Oracle Enterprise Manager Cloud Control (EM Cloud Control):** Use this GUI tool to perform database administration tasks on several targets (database instances, listeners, and so on) at the same time.
- **Oracle Management Cloud Database Express (OMX):** Use this GUI to perform database administration tasks. OMX is a replacement for EM Express with a subset of functionality in the initial release.
- **Others:** Many specialized utilities are used to assist with database administration. The ones used in this course may include Listener Control (`lsnrctl`), Oracle Net Configuration Assistant (`netca`), Oracle Net Manager (`netmgr`), ADR Command Interpreter (`adcri`), SQL*Loader (`sqlldr`), Oracle Data Pump Import (`impdp`), and Oracle Data Pump Export (`expdp`). This list does not cover all the utilities available.

Database Tool Choices

Topic	SQL*Plus	SQL Developer	SQLcl	DBCA	EM Database Express	EM Cloud Control	Oracle Universal Installer
Create a CDB or PDB	Yes	Yes (PDB only)	Yes	Yes	Yes (PDB only)	Yes (PDB only)	Yes
Explore CDB instance, architecture, and PDBs	Yes	Yes	Yes	No	Yes	Yes	No

The table shows you at a glance which tool can be used to perform which tasks.

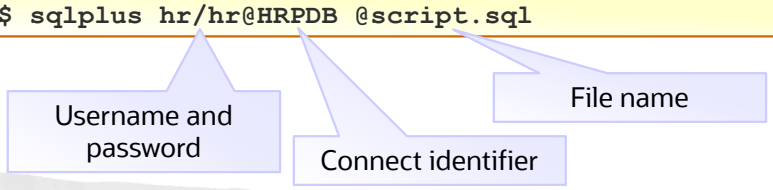
SQL*Plus

- **Example 1:** From a command line, you can start SQL*Plus, log in, and show the user that you're logged in as:

```
$ sqlplus / as sysdba
...
SQL> show user
USER is "SYS"
```

- **Example 2:** Call a SQL script from the command line:

```
$ sqlplus hr/hr@HRPDB @script.sql
```



SQL*Plus

SQL*Plus is a command-line program that you use to submit SQL and PL/SQL statements to an Oracle database. SQL*Plus is installed with Oracle Database and is located in the `$ORACLE_HOME/bin` directory. You can start SQL*Plus from the command line or from the Start menu on a Windows client. Use the SQL*Plus command-line interface to execute SQL*Plus, SQL, and PL/SQL commands to perform the following:

- Enter, edit, run, store, retrieve, and save SQL commands and PL/SQL blocks
- Format, calculate, store, and print query results
- List column definitions for any table
- Send messages to and accept responses from an end user
- Perform database administration

Calling a SQL Script from SQL*Plus

When calling a SQL script file from within SQL*Plus, you have the following options:

- **Option 1:** Call the script from the command line when you first invoke SQL*Plus:

```
$ sqlplus hr/hr@HRPDB @script.sql
```

- **Option 2:** Call the script from inside a SQL*Plus session simply by using the "@" operator:

```
SQL> @script.sql
```

When a script is saved from SQL*Plus by using the `SAVE` command, the `.sql` extension is automatically supplied. You can then execute the script without supplying the extension at execution time, for example:

```
SQL> @script
```

Calling SQL*Plus from a Shell Script

You can call SQL*Plus from a shell script or BAT file by invoking `sqlplus` and using the operating system scripting syntax for passing parameters. For example, suppose you have the following shell script:

```
# Name of this file: batch_sqlplus.sh

# Count employees and give raise.

sqlplus hr/hr <<EOF

select count(*) from employees;

update employees set salary = salary*1.10;

commit;

quit

EOF
```

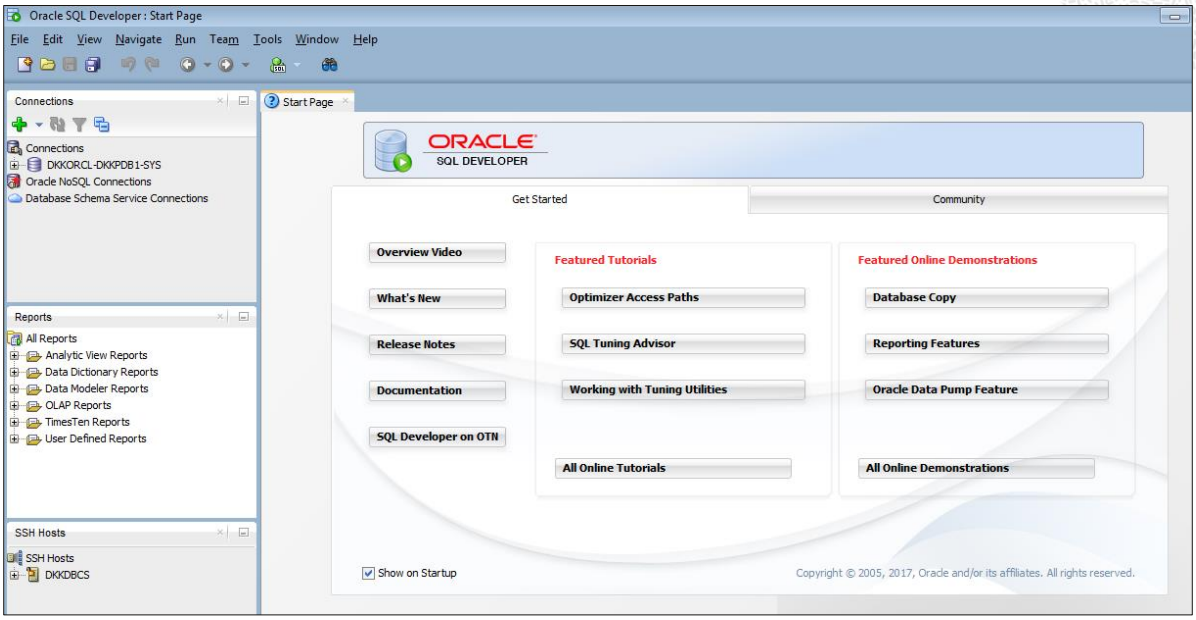
You can call that shell script from the command line:

```
$ ./batch_sqlplus.sh
```

For More Information

To learn how to start SQL*Plus, see "Starting SQL Plus" in *SQL*Plus Users Guide and Reference*.

Oracle SQL Developer



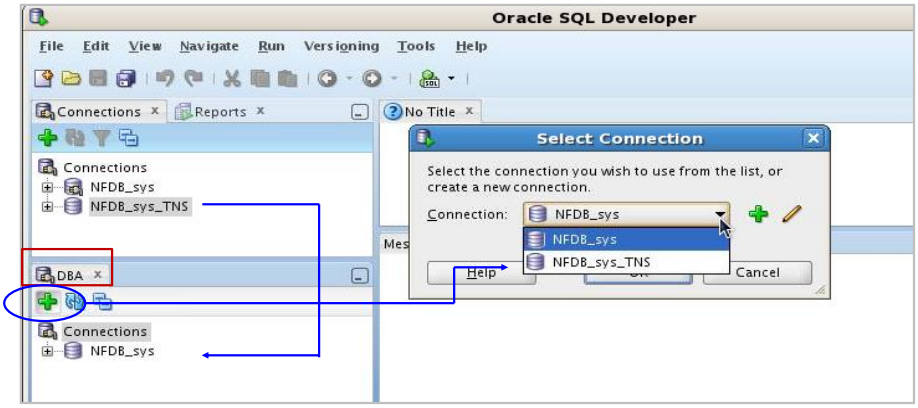
Oracle SQL Developer (SQL Developer) is a graphical tool for database developers and DBAs and gets installed with Oracle Database. You can choose to display several different panes in the SQL Developer interface, such as a Connections and DBA pane. You use the former to define connections to databases and use the latter to perform DBA operations. You should add connections only for which the associated database user has DBA privileges or at least privileges for the desired DBA navigator operations on the specified database.

Some developer-type operations that you can perform with SQL Developer include:

- Develop scripts in both the SQL and PL/SQL languages
- Browse database objects
- Run SQL statements and SQL scripts
- Edit and debug PL/SQL statements

Oracle SQL Developer: Connections

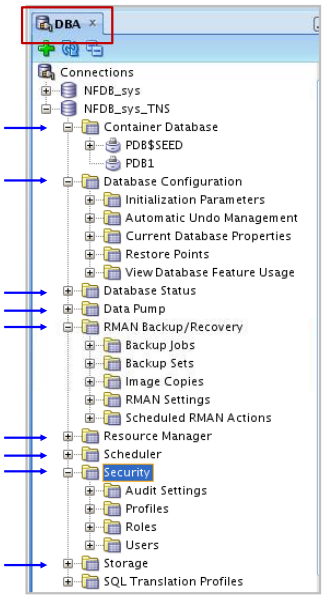
- Perform DBA operations in the DBA navigator by using DBA connections:



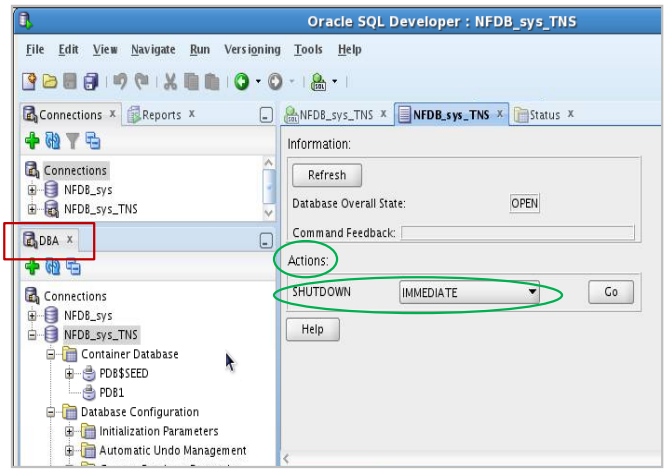
Oracle SQL Developer is a tool that allows stand-alone graphical browsing and development of database schema objects, as well as execution of database administrative tasks.
SQL Developer enables users with database administrator privileges to view and edit certain information relevant to DBAs and perform DBA operations. To perform DBA operations, use the DBA navigator, which is similar to the Connections navigator in that it has nodes for all defined database connections. If the DBA navigator is not visible, select View, then DBA. You should add only connections for which the associated database user has DBA privileges, or at least privileges for the desired DBA navigator operations on the specified database.

Oracle SQL Developer: DBA Actions

Using DBA features through DBA navigator



Performing DBA actions



The DBA operations that can be performed are the following:

- Database startup/shutdown
- Database configuration: Initialization Parameters, Automatic Undo Management, Current Database Properties, Restore Points, View Database Feature Usage
- Database status view
- Data Pump export and import jobs
- RMAN backup/recovery actions
- Resource Manager configuration
- Scheduler setting
- Security configuration like audit settings, profiles, roles, and users
- Storage configuration for archive logs, control files, data files, redo log groups, tablespaces, and temporary tablespace groups

Database Configuration Assistant (DBCA)

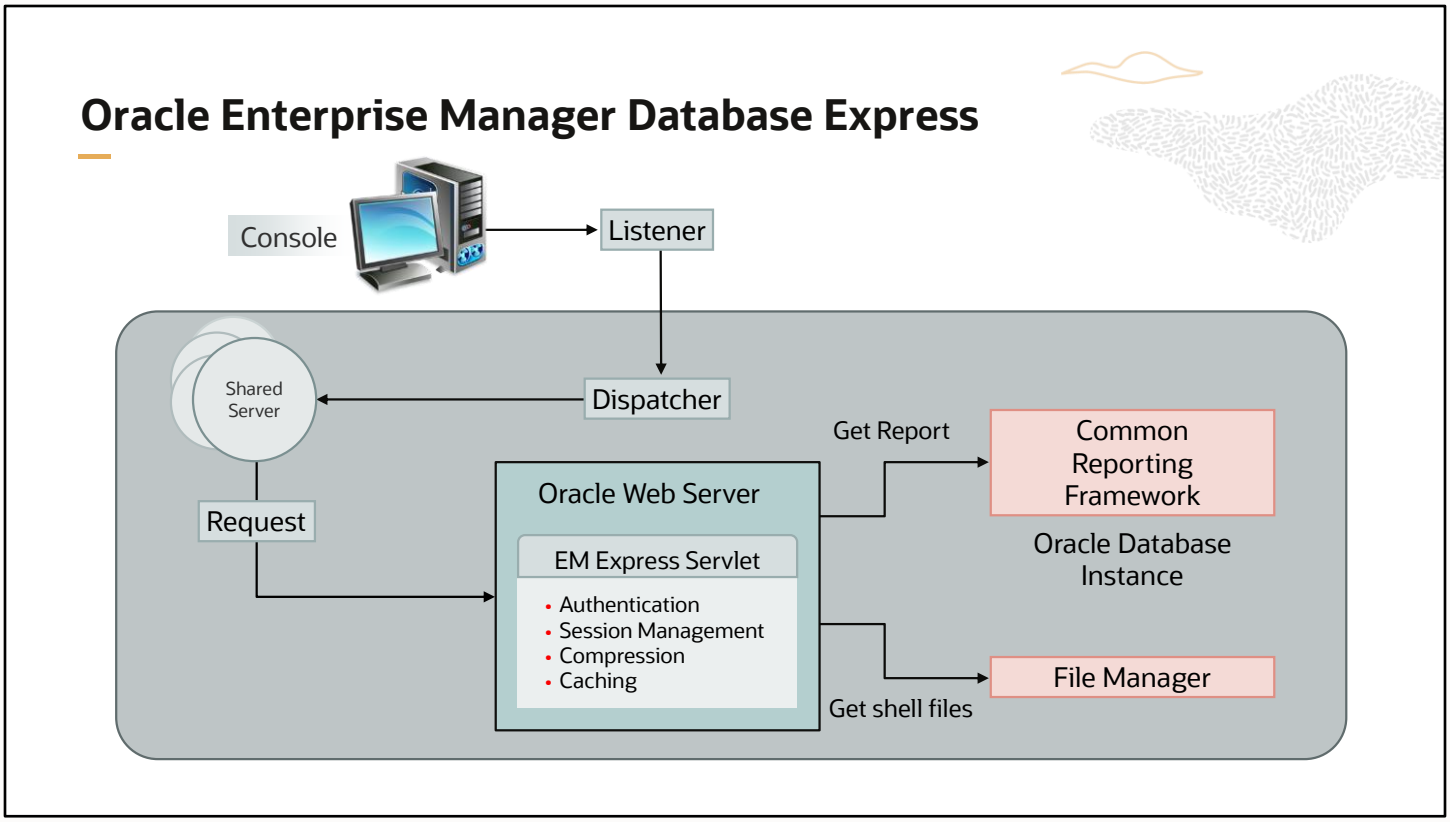
- DBCA is a tool for creating and configuring an Oracle database.
- DBCA has two modes:
  - Interactive: Provides a graphical interface and guided workflow
  - Noninteractive/silent: Uses command-line arguments, a response file, or both
- DBCA can be launched by the Oracle Universal Installer or invoked after the software is installed.

Database Configuration Assistant (DBCA) is a tool that you can use to do the following:

- Create databases (CDBs and non-CDBs)
- Configure existing databases
- Delete databases
- Manage templates
- Manage pluggable databases
- Manage Oracle RAC database instances

DBCA can be launched by the Oracle Universal Installer (OUI), depending upon the type of install that you select. You can also launch DBCA as a stand-alone tool at any time after Oracle Database installation.

See *Oracle Database Administrator's Guide and Oracle Database 2 Day DBA* for detailed information.



Oracle Enterprise Manager Database Express

Oracle Enterprise Manager Database Express (EM Express) is a lightweight tool that you can use to manage a CDB and all of its PDBs (except the seed PDB). It provides an out-of-the-box browser-based management solution, performance monitoring, configuration management, administration, diagnostics, and tuning.

Architecture

EM Express uses a web-based console, communicating with the built-in web server available in XML DB.

As requests from the console are processed, the EM Express servlet handles the requests, including authentication, session management, compression, and caching. The servlet passes requests for reports to the Common Reporting Framework and actions requiring shell files to the File Manager. This architecture is illustrated in the slide.

EM Express is available only when the database is open. This means that Enterprise Manager Database Express cannot be used to start the database instance. Other operations that require that the database change state, such as enable or disable ARCHIVELOG mode, are also not available in EM Express.

EM Express is configurable with a single click in Database Configuration Assistant (DBCA).

EM Express is a servlet built on top of Oracle XML DB. The Oracle XML DB default wallet has a self-signed certificate, and some existing browsers consider self-signed certificates as untrusted because they are not signed by a trusted CA (certificate authority). However, the self-signed certificate is still secure, as it ensures that the traffic is encrypted between the Oracle XML DB server and the client (browser). Therefore, enter a security exception for the EM Express URL in your web browser.

Requirements

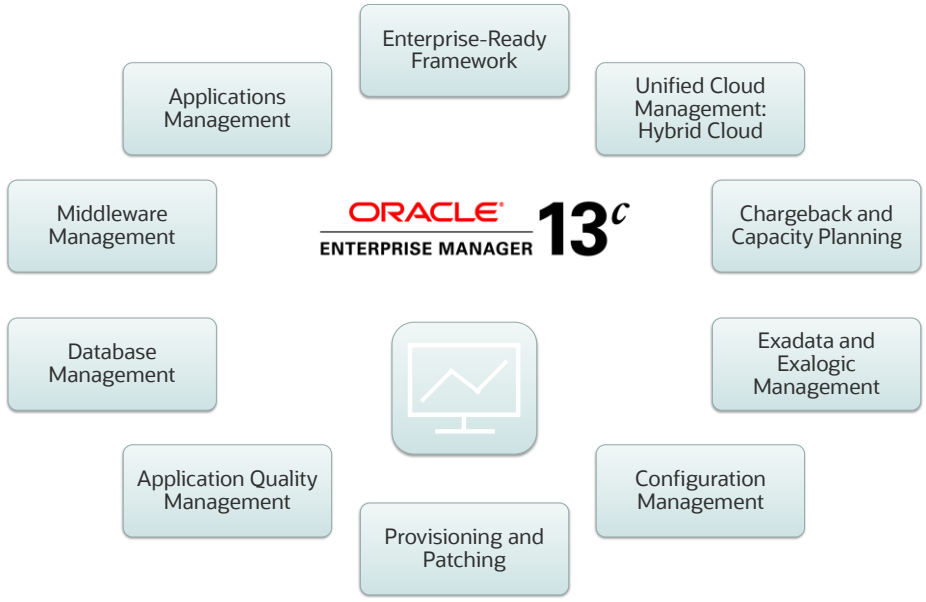
The following are required for EM Express:

- XMLDB components must be installed on the Oracle Database server. All Oracle Databases of version 12.1.0 or higher have XMLDB installed.
- The web browser must have the Flash plug-in installed because EM Express uses Shockwave Flash (SWF) files.

Starting EM Express for CDBs and PDBs

EM Express uses a global HTTPS port to connect to and manage non-CDBs, CDBs, and PDBs. Starting with Oracle Database 12*c* Release 2, you can set a global port that enables Enterprise Manager Database Express access to the CDB and all PDBs on that single port.

Enterprise Manager Cloud Control 13*c* Features



Oracle Enterprise Manager Cloud Control (EM Cloud Control) is Oracle's on-premises management platform, providing a single location for managing all your Oracle deployments, whether they be in your data centers or in Oracle Cloud. Through deep integration with Oracle's product stack, EM Cloud Control provides management and automation support for Oracle applications, databases, middleware, hardware, and engineered systems.

Key objectives in the design of Enterprise Manager Cloud Control 13*c* include:

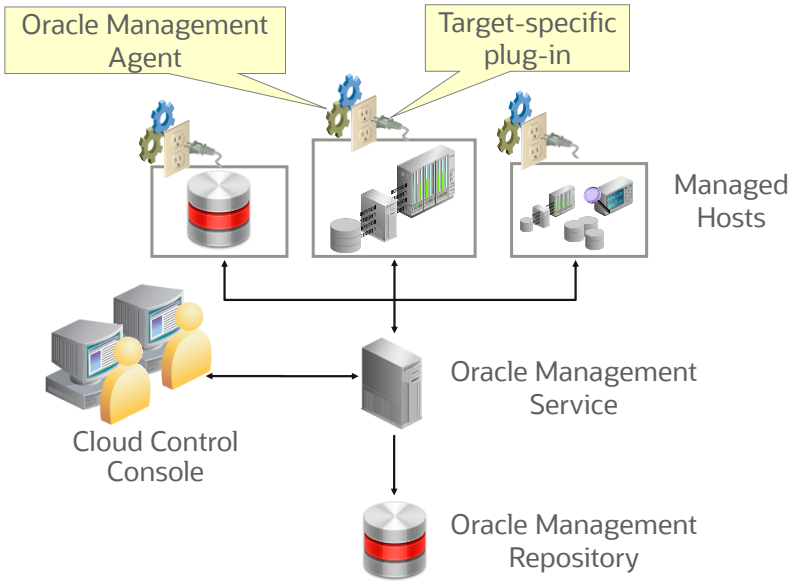
- Providing a complete integrated cloud management solution for a combination of on-premises cloud configurations and Oracle Cloud Services solutions (Hybrid Cloud)
- Delivering enhanced engineered systems management
- Enhancing middleware and database management
- Maintaining a robust, cloud-scale platform

Enterprise Manager Cloud Control 13*c* includes the following features:

- **Enterprise-Ready Framework:** Provides modular and extensible architecture, self-updateable entities, centralized incident management, in-context diagnostics management, as well as flexible job scheduling and security sub-systems
- **Cloud Management:** Provides complete cloud life cycle management for both on-premises clouds and PaaS services on Oracle Cloud
- **Chargeback and Capacity Planning:** Provides chargeback based on target types and uses Automatic Workload Repository (AWR) Warehouse to consolidate AWR reports from multiple databases across the enterprise

- **Exadata and Exalogic Management:** Provides an integrated view of the hardware and software in an Exadata machine and complete life cycle management for Exalogic systems
- **Configuration and Management:** Provides an integrated set of tools, agent-less discovery, integration with My Oracle Support, and custom configuration capabilities
- **Provisioning and Patching:** Provides profiles for provisioning known configurations, user-defined deployment procedures, and a Software Library integrated with self-updating capabilities
- **Application and Quality Management:** Provides Database Replay, Application Server Replay, Real Application Testing integrated with Data Masking, and test database management that includes Application Data Model
- **Database Management:** Provides management of Oracle Database systems, including performance management and change life cycle management
- **Middleware Management:** Provides complete management of Middleware systems
- **Applications Management:** Provides management of Fusion Applications

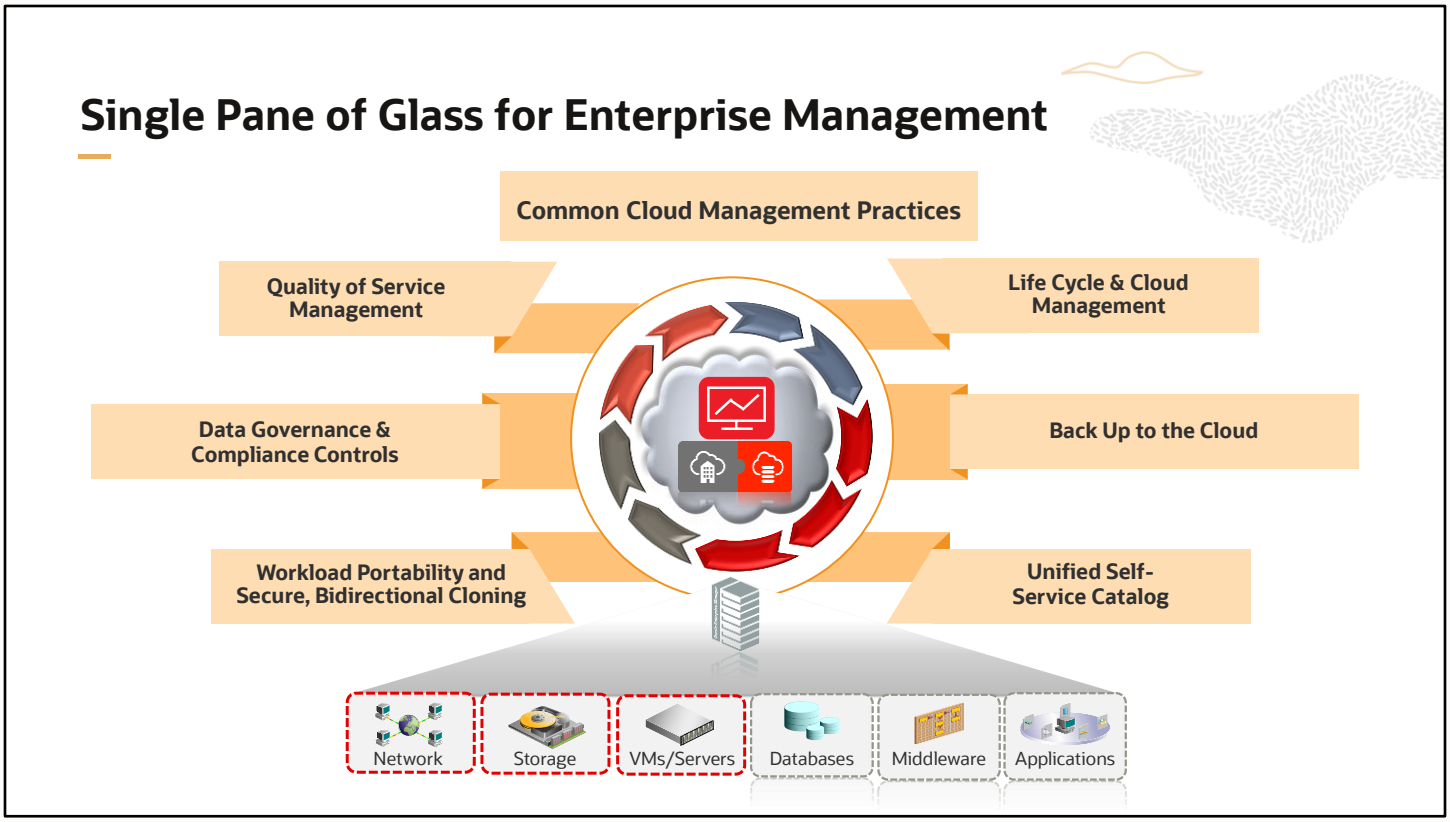
Oracle Enterprise Manager Component Overview



Oracle Enterprise Manager Cloud Control is composed of four main components as illustrated in the slide:

- Oracle Management Repository (OMR)
- Oracle Management Service (OMS)
- Oracle Management Agent (OMA or agent) with target-specific plug-ins
- Cloud Control Console

The Oracle Management Agent runs on hosts, gathering metric data about those host environments as well as using plug-ins to monitor availability, configuration, and performance, and to manage targets running on the host. The agents communicate with the Oracle Management Service to upload metric data collected by them and their plug-ins. In turn, the OMS stores the data it collects in the Oracle Management Repository where it can be accessed by the OMS for automated and manual reporting and monitoring. The OMS also communicates with the agents to orchestrate the management of their monitored targets. As well as coordinating the agents, the OMS runs the Cloud Control Console web pages that are used by administrators and users to report on, monitor, and manage the computing environment that is visible to Cloud Control via the agents and their plug-ins.

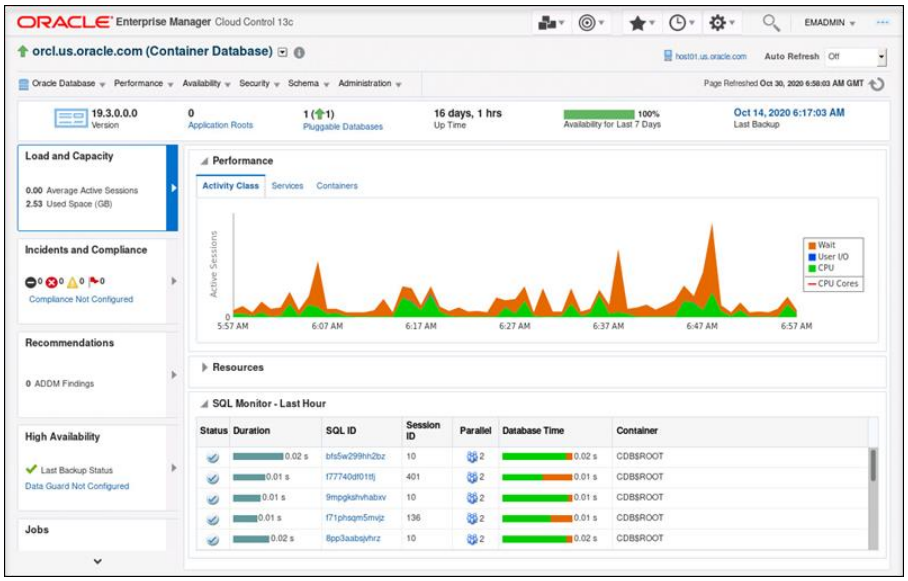


One Management Tool to Oversee Them All

Enterprise Manager Cloud Control 13c has capabilities to intelligently manage traditional and cloud-based services, mitigating the need to use multiple management and monitoring tools for what were previously two disparate environments.

- Complete solution for management of the Oracle stack, including engineered systems, with real-time integration of Oracle's knowledge base with customer environments
- End-to-end performance management and automation
- Integrated Ops Center functionality to monitor and manage both hardware and software from a single interface
- Common management practices applicable to on-premises targets and Oracle Cloud targets
- Quality of Service (QoS) management to ensure delivery of the best service possible to internal and external customers
- Life cycle and cloud management for simplified provisioning and patching of applications and platforms
- Data governance and compliance controls for conforming with internal and external standards and requirements
- Ability to back up to the cloud to leverage the Oracle Cloud capacity
- Hybrid cloud option to move workloads and clone targets between on-premises and Oracle Cloud

Oracle Enterprise Manager Database Management



Database management involves the monitoring, administration, and maintenance of the databases and database groups in your enterprise. With Enterprise Manager, you receive a complete set of integrated features for managing Oracle Database, which enables you to:

- Perform day-to-day tasks and run repetitive jobs
- Detect problems and use the available guided resolution workflows
- Manage a single database or thousands of database instances via scalability
- Lead the industry in ease of deployment and use with an intuitive management product

Using Oracle Enterprise Manager Cloud Control, you can keep your Oracle Databases available and running efficiently.

You can use the available administration tools to manage database objects and initiate database operations inside an Oracle Database.

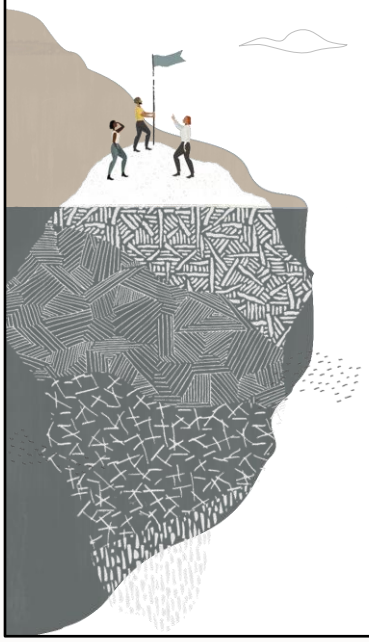
The following list provides some of the database administration tasks you can perform:

- Allocate system storage and plan future storage requirements.
- Use the Oracle Scheduler to control when and where various tasks occur in the database environment.
- Migrate your database storage to use Oracle Automatic Storage Management.

Oracle Enterprise Manager Database Details Page (continued)

- Use the Database Resource Manager to control the distribution of resources among various sessions by controlling the execution schedule inside the database.
- Create and manage primary database objects such as tables, views, and indexes.
- Perform administration tasks for Oracle XML DB, such as viewing or editing parameters for the Oracle XML DB configuration file.
- Create, manage, and perform specific actions against materialized views, which are schema objects that can be used to summarize, compute, replicate, and distribute data.

Summary



- Connect to Oracle Database
- Describe the tools used to access Oracle Database

ORACLE
University

Creating an Oracle Database by Using DBCA

Objectives

- Create a database by using the Database Configuration Assistant (DBCA)
- Generate database creation scripts by using DBCA

Planning the Database

- As a DBA, you must plan:
  - The logical storage structure of the database and its physical implementation:
    - What type of storage is being used?
    - How many data files will you need? (Plan for growth)
    - How many tablespaces will you use?
    - What types of information will be stored?
    - Are there any special storage requirements due to type or size?
  - Overall database design
  - Database backup strategy

It is important to plan how the logical storage structure of the database will affect system performance and various database management operations. For example, before creating any tablespaces for your database, you should know how many data files will make up the tablespace, what type of information will be stored in each tablespace, and where the data files will be physically stored. Information such as the availability of network attached storage (NAS) and the bandwidth for the private storage network is important. If storage area networks (SAN) are going to be used, knowing how the logical volumes are configured and the stripe size is useful.

When planning the overall logical storage of the database structure, take into account the effects that this structure will have when the database is actually created and running. You may have database objects that have special storage requirements due to type or size.

In distributed database environments, this planning stage is extremely important. The physical location of frequently accessed data dramatically affects application performance.

During the planning stage, develop a backup strategy for the database. You can alter the logical storage structure or design of the database to improve backup efficiency. Backup strategies are introduced in a later lesson.

Choosing a Database Template

- General purpose or transaction processing:
  - Online transaction processing (OLTP) system, for example, a retail billing system for a software house or a nursery
- Custom:
  - Multipurpose database (perhaps combined OLTP and data warehouse functionality)
- Data warehouse:
  - Research and marketing data
  - State or federal tax payments
  - Professional licensing (doctors, nurses, and so on)

Different types of databases have their own specific instance and storage requirements. Your Oracle database software includes templates for the creation of these different types of databases. Characteristics of these examples are the following:

- **General purpose:** For general purpose or transaction processing usage such as working with transactions and storing them for a medium length of time
- **Custom:** For customized databases that do not fit into the general purpose or data warehouse template
- **Data warehouse:** For storing data for long periods and retrieving them in read operations

Choosing the Appropriate Character Set

- The character set is chosen at the time of database creation. Choose the character set that best meets your business requirements now and in the future because it can be difficult to change character sets later on.
- The Oracle database supports different classes of character-encoding schemes:
  - Single-byte character sets
    - 7-bit
    - 8-bit
  - Multibyte character sets, including Unicode
- In general, Unicode is recommended because it is the most flexible character set.

When computer systems process characters, they use numeric codes instead of the graphical representation of the character. An encoded character set maps numeric codes to characters that a computer or terminal can display and receive. Different character sets support different character repertoires. Because character sets are typically based on a particular writing script, they can support more than one language. However, script-based character sets are restricted in the sense that they are limited to groups of languages based on similar scripts. Universal character sets encompass most major scripts of the modern world and provide a more useful solution to multilingual support. For information about the Unicode standards, see the web site at http:www.unicode.org.

The Oracle database supports three classes of encoding schemes: Single-byte, Varying-width multibyte, and Universal. Choose the correct character set that best meets your business requirements now and in the future because it can be difficult to change character sets later on. For best performance, choose a character set that avoids character set conversion and uses the most efficient encoding for the languages desired. Single-byte character sets result in better performance than multibyte character sets, and they also are the most efficient in terms of space requirements. However, single-byte character sets limit how many languages you can support. To choose your correct database character set, evaluate your current and future business requirements, as well as technical requirements (for example, the XML and Java standards require Unicode). In general, Oracle recommends the use of Unicode for all new databases, because it is the most flexible character set and avoids future conversions.

Single-Byte Character Sets

In a single-byte character set, each character occupies one byte. Single-byte 7-bit encoding schemes can define up to 128 (27) characters; single-byte 8-bit encoding schemes can define up to 256 (28) characters.

Examples of Single-Byte Schemes

- 7-bit character set:
  - American Standard Code for Information Interchange (ASCII) 7-bit American (US7ASCII)
- 8-bit character set:
  - International Organization for Standards (ISO) 8859-1 West European (WE8ISO8859P1)
  - DEC 8-bit West European (WE8DEC)
  - Extended Binary Coded Decimal Interchange Code (EBCDIC) Code Page 1144 8-bit Italian (I8EBCDIC1144)

Multibyte Character Sets

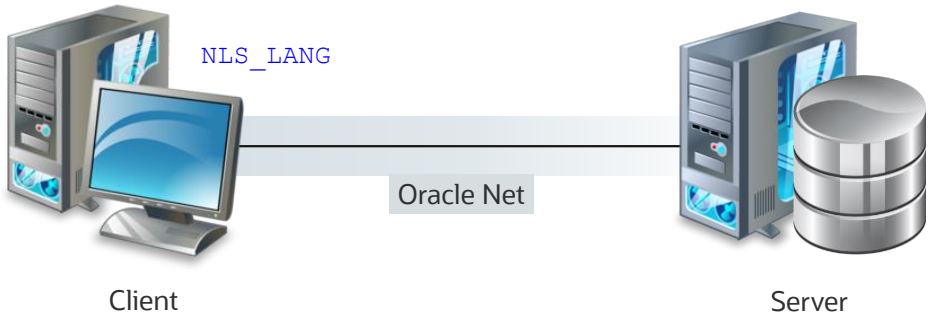
- A varying-width multibyte character set is represented by one or more bytes per character. Multibyte character sets are commonly used for Asian language support. Some multibyte encoding schemes use the value of the most significant bit to indicate whether a byte represents a single byte or is part of a series of bytes representing a character. However, other character-encoding schemes differentiate single-byte from multibyte characters. A shift-out control code, sent by a device, indicates that any successive bytes are double-byte characters until a shift-in code is encountered. Shift-sensitive encoding schemes are used primarily on IBM platforms.
- Unicode is a universal encoded character set that enables information from any language to be stored using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language.
- The Unicode standard has been adopted by many software and hardware vendors. Many operating systems and browsers now support Unicode. Unicode is required by standards such as XML, Java, JavaScript, LDAP, and WML. It is also synchronized with the ISO/IEC 10646 standard.

Examples of Varying-Width Multibyte Schemes

- Shift-JIS 16-bit Japanese (JA16SJIS)
- MS Windows Code Page 950 with Hong Kong Supplementary Character Set HKSCS-2001 (ZHT16HKSCS)
- Unicode 4.0 UTF-8 Universal character set (AL32UTF8) - a variable-width type of encoding and also a strict superset of ASCII
- Unicode (ALl16UTF16 – a 16-bit encoding of Unicode that is used by both Microsoft Windows 2000 and Windows XP

How are character sets used?

- Oracle Net compares the client NLS_LANG setting to the character set on the server.
- If needed, conversion occurs automatically and transparently.



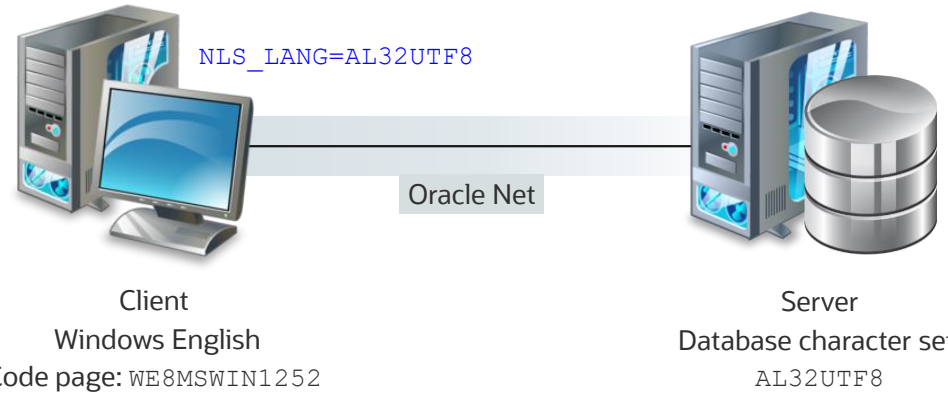
The NLS_LANG parameter defines a client terminal's character-encoding scheme. Different clients can use different encoding schemes. Data passed between the client and the server is converted automatically between the two encoding schemes. The database's encoding scheme should be a superset, or equivalent, of all the clients' encoding schemes. The conversion is transparent to the client application.

When the database character set and the client character set are the same, the database assumes that the data being sent or received is of the same character set, so no validations or conversions are performed.

Character set conversion may be required in a client/server environment, if a client application resides on a different platform than the server and if the platforms do not use the same character-encoding schemes. Character data passed between the client and the server must be converted between the two encoding schemes. Character conversion occurs automatically and transparently through Oracle Net.

Setting NLS_LANG Correctly on the Client

- No conversion occurs, because it does not seem to be required.
- Invalid data is entered into the database.



Invalid data usually enters a database when the NLS_LANG parameter is not set properly on the client. The NLS_LANG value should reflect the encoding of the incoming data.

When the NLS_LANG parameter is set properly, the database can automatically convert incoming data from the client operating system.

When the NLS_LANG parameter is not set properly, the data entering the database is not converted properly.

For example, suppose that the database character set is AL32UTF8, the client is an English Windows operating system (code page: WE8MSWIN1252), and the NLS_LANG setting on the client is AL32UTF8. Data entering the database is encoded in WE8MSWIN1252 and is not converted to AL32UTF8 data because the NLS_LANG setting on the client matches the database character set. The Oracle database server assumes that no conversion is necessary, and invalid data is entered into the database.

Using the Database Configuration Assistant

Task	Description
Create a database	Create and configure a new database
Create and manage database design templates	Create an XML file containing information required to create a new database
Delete a database	Shut down the database instance and delete all database files
Manage pluggable databases in a multitenant architecture	Create, delete, and unplug PDBs
Change the configuration of a database or PDB	Configure options and security settings

Oracle Database Configuration Assistant (DBCA) is a tool for creating and configuring an Oracle database. DBCA can be launched by the Oracle Universal Installer (OUI), depending upon the type of install that you select. You can also launch DBCA as a stand-alone tool at any time after Oracle Database installation.

You can run DBCA in interactive mode or noninteractive/silent mode. Interactive mode provides a graphical interface and guided workflow for creating and configuring a database.

You can use DBCA to create a database from a template supplied by Oracle or from a template that you create. A DBCA template is an XML file that contains information required to create a database. Select the template suited to the type of workload your database will support. If you are not sure which to choose, then use the "General purpose OR online transaction processing" template. You can also create custom templates to meet your specific workload requirements.

The information in templates includes database options, initialization parameters, and storage attributes (for data files, tablespaces, control files, and online redo log files).

Templates can be used just like scripts, but they are more powerful than scripts because you have the option of duplicating a database. Duplication saves time because you copy the files of an existing database, referred to as a seed database, to the correct locations.

With a multitenant container database (CDB), you can use DBCA to perform pluggable database (PDB) operations in the CDB. You can create, delete, and unplug a PDB.

You can use DBCA to change the configuration of an existing database. For example, you can make configuration changes such as:

- Add database options that were not previously configured
- Change default security settings
- Change the server mode from dedicated to shared or the reverse

Using DBCA in Silent Mode

- Create a new database named `ORCL` using the General Purpose template:

```
$ORACLE_HOME/bin/dbca -silent -createDatabase
-templateName General_Purpose.dbc -gdbname ORCL
-sid ORCL -createAsContainerDatabase true
-numberOfPDBs 1 -pdbName pdb1 -useLocalUndoForPDBs true
-responseFile NO_VALUE -characterSet AL32UTF8
-totalMemory 1800 -sysPassword Welcome_1
-systemPassword Welcome_1 -pdbAdminPassword Welcome_1
-emConfiguration DBEXPRESS -dbsnmpPassword Welcome_1
-emExpressPort 5500 -enableArchive true
-recoveryAreaDestination /u03/app/oracle/fast_recovery_area
-recoveryAreaSize 15000 -datafileDestination /u02/app/oracle/oradata
```

- Delete a database named `ORCL`:

```
$ORACLE_HOME/bin/dbca -silent -deleteDatabase -sourceDB ORCL -sid ORCL
-sysPassword Welcome_1
```

You can also create a database using the noninteractive/silent mode of DBCA. Noninteractive/silent mode enables you to script database creation. You can run DBCA in noninteractive/silent mode by specifying command-line arguments, a response file, or both.

Specify true to create a CDB. Specifying false is not supported starting with Oracle Database Release 20.3.

When true is specified, the following additional parameters are optional:

-numberOfPDBs: Number of PDBs to create. The default is 0 (zero).

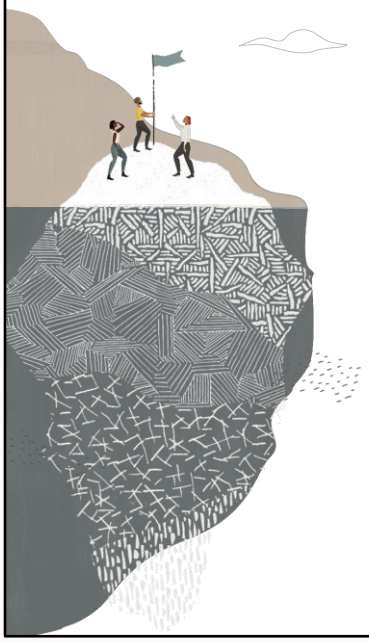
-pdbName: Base name of each PDB. A number is appended to each name if -numberOfPDBs is greater than 1. This parameter must be specified if -numberOfPDBs is greater than 0 (zero).

-useLocalUndoForPDBs {true | false}: Specify whether local undo should be used for the PDBs.

-pdbAdminPassword: PDB administrator password

See the *Oracle Database Administrator's Guide* for detailed information about the syntax and options for the Database Configuration Assistant (DBCA) silent mode commands.

Summary



- Create a database by using the Database Configuration Assistant (DBCA)
- Generate database creation scripts by using DBCA

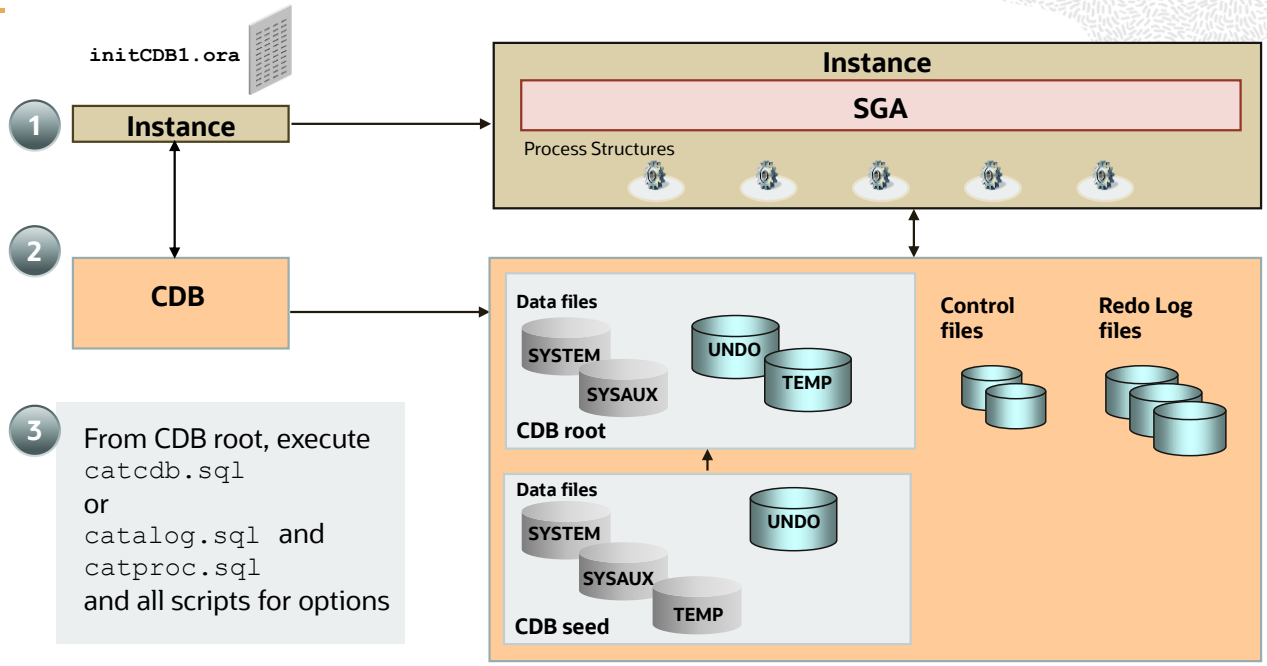
ORACLE
University

Creating an Oracle Database by Using a SQL Command

Objectives

Create a container database (CDB) by using the `CREATE DATABASE` command

Creating a Container Database (CDB)



Use the `CREATE DATABASE` statement to create a database, making it available for general use.

This statement erases all data in any specified data files that already exist in order to prepare them for initial database use. If you use the statement on an existing database, then all data in the data files is lost.

To create a database, you must have the SYSDBA system privilege. An initialization parameter file with the name of the database to be created must be available, and you must be in STARTUP NOMOUNT mode.

Starting with Oracle Database 21c, the `ENABLE_PLUGGABLE_DATABASE` initialization parameter is set to TRUE by default. If you set the `ENABLE_PLUGGABLE_DATABASE` initialization parameter to FALSE, the command will fail.

The `CREATE DATABASE enable_pluggable_database` statement creates a CDB that contains a root and a seed container. You then create PDBs in the CDB by using the `CREATE PLUGGABLE DATABASE` statement.

The operation creates the control files during the mount phase and the redo log files and CDB root data files during the open phase. The CDB root data files are used for the `SYSTEM` tablespace containing the Oracle-supplied metadata and data dictionary, the `SYSAUX` tablespace for the Automatic Workload Repository (AWR), and the undo tablespace.

It also creates the CDB seed with its own data files used for the `SYSAUX`, `SYSTEM`, and undo tablespaces. You may use the `SEED FILE_NAME_CONVERT` clause to define the data files location of the CDB seed pluggable database. The clause creates the CDB seed. The CDB seed data files can be used as templates for future PDBs creation. If you omit this clause, Oracle Managed Files (OMF) determines the names and locations of the CDB seed's files.

Creating a CDB by Using a SQL Command: Example

1. Start up the instance :
   a. Set `ORACLE_SID=CDB1`.
   b. Create the `initCDB1.ora` file and set parameters:
      - `CONTROL_FILES` to CDB control file names
      - `DB_NAME` to a CDB name
      - `ENABLE_PLUGGABLE_DATABASE` to `TRUE`

```
SQL> CONNECT / AS SYSDBA
SQL> STARTUP NOMOUNT
```

2. Create the database:

```
SQL> CREATE DATABASE cdb1 ENABLE PLUGGABLE DATABASE …
            SEED FILE_NAME_CONVERT = ('/oracle/dbs','/oracle/seed');
```

   ➔ `CDB$ROOT + PDB$SEED` created

3. Execute the `$ORACLE_HOME/rdbms/admin/catcdb.sql` SQL script.

Below are the detailed steps to create a new CDB using SQL*Plus.

1. Before starting the instance, create an initialization parameter file with the parameters: `DB_NAME`, `CONTROL_FILES` if OMF is not used, and `DB_BLOCK_SIZE`. The global database name of the CDB root is the global database name of the CDB. You can also use the `MAX_PDBS` parameter to limit the number of PDBs in the CDB. Set the `ORACLE_SID` environment variable. Launch SQL*Plus, connect as an OS authenticated user belonging to the DBA OS group, and execute the `STARTUP NOMOUNT` command. If you are using Oracle Automatic Storage Management (ASM) storage to manage your disk storage, then you must start the Oracle ASM instance and configure your disk groups before performing the next steps.
2. Use the `CREATE DATABASE` command with the `ENABLE PLUGGABLE DATABASE` clause to create a CDB. The command creates the CDB root and the CDB seed. You can use the `SEED FILE_NAME_CONVERT` clause to specify the location of the CDB seed's files. If you omit the clause, OMF determines the names and locations of the CDB seed's files. `FILE_NAME_CONVERT` specifies the source directory of the CDB root data files and the target CDB seed directory.

   Omit the `SEED FILE_NAME_CONVERT` clause if you use the `PDB_FILE_NAME_CONVERT` initialization parameter that maps names of the CDB root data files to the CDB seed data files. The directories must exist. The character set defined is the single one for the CDB.
3. Run the `catcdb.sql` SQL script to build views on the data dictionary tables and install standard PL/SQL packages in the CDB root. You can also execute the `catalog.sql` and `catproc.sql` SQL scripts and all other SQL scripts related to the options installed.

Using the SEED FILE_NAME_CONVERT Clause

```
SQL> CREATE DATABASE cdb1
     USER SYS IDENTIFIED BY p1 USER SYSTEM IDENTIFIED BY p2
     LOGFILE GROUP 1 ('/u01/app/oradata/CDB1/redo1a.log',
                      '/u02/app/oradata/CDB1/redo1b.log') SIZE 100M,
             GROUP 2 ('/u01/app/oradata/CDB1/redo2a.log',
                      '/u02/app/oradata/CDB1/redo2b.log') SIZE 100M
     CHARACTER SET AL32UTF8 NATIONAL CHARACTER SET AL16UTF16
     EXTENT MANAGEMENT LOCAL DATAFILE
                     '/u01/app/oradata/CDB1/system01.dbf' SIZE 325M
     SYSAUX DATAFILE  '/u01/app/oradata/CDB1/sysaux01.dbf' SIZE 325M
     DEFAULT TEMPORARY TABLESPACE tempts1
            TEMPFILE '/u01/app/oradata/CDB1/temp01.dbf' SIZE 20M
     UNDO TABLESPACE undotbs
             DATAFILE '/u01/app/oradata/CDB1/undotbs01.dbf' SIZE 200M
     ENABLE PLUGGABLE DATABASE
     SEED
     FILE_NAME_CONVERT =('/u01/app/oradata/CDB1','/u01/app/oradata/CDB1/seed');
```

One way to declare the directory for the PDB seed data files is to use the `SEED FILE_NAME_CONVERT` clause. The `FILE_NAME_CONVERT` clause specifies the source directory of the CDB root data files and the target CDB seed directory.

The `ENABLE PLUGGABLE DATABASE` clause indicates that a CDB is being created. The CDB will contain a root (CDB$ROOT) and a seed (PDB$SEED). The `FILE_NAME_CONVERT` clause specifies that names of files for the seed will be generated by replacing `/u01/app/oradata/CDB!` in the names of files associated with the root with `/u01/app/oradata/CBD1/seed`.

The `/u01/app/oradata/CDB1` root directory and the `/u01/app/oradata/CDB1/seed` seed directory must exist.

Using the ENABLE PLUGGABLE DATABASE Clause

- Without SEED FILE_NAME_CONVERT:
  - OMF: DB_CREATE_FILE_DEST='/u01/app/oradata'

```
SQL> CONNECT / AS SYSDBA
SQL> STARTUP NOMOUNT
SQL> CREATE DATABASE cdb2
      USER SYS IDENTIFIED BY p1 USER SYSTEM IDENTIFIED BY p2
      EXTENT MANAGEMENT LOCAL
      DEFAULT TEMPORARY TABLESPACE temp
      UNDO TABLESPACE undotbs
      DEFAULT TABLESPACE users
      ENABLE PLUGGABLE DATABASE;
```

  - Or initialization parameter: PDB_FILE_NAME_CONVERT = '/u01/app/oradata/CDB1','/u01/app/oradata/CDB1/seed'

Oracle Managed Files

If you do not use explicit data file names, use Oracle Managed Files (OMF):

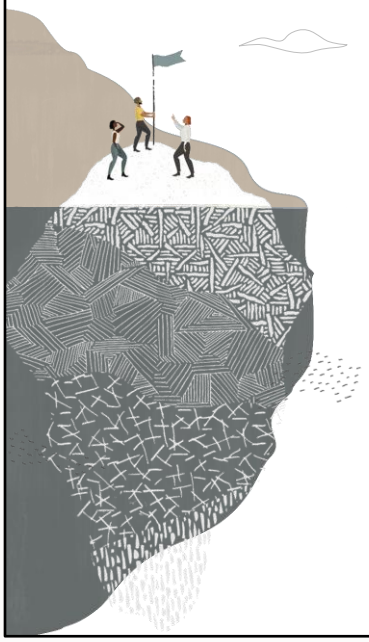
- Set the DB_CREATE_FILE_DEST initialization parameter with the value of the destination directory of the data files of the SYSTEM, SYSAUX, UNDO, and USERS tablespaces specified in the statement. Oracle chooses default sizes and properties for all data files, control files, and redo log files.
- The /u01/app/oradata directory must exist.

PDB_FILE_NAME_CONVERT Instance Parameter

If you do not use the SEED FILE_NAME_CONVERT clause, use an initialization parameter:

- The PDB_FILE_NAME_CONVERT initialization parameter maps names of existing files (the root data files in your case) to new file names (the seed data files in this case).
- In the example, both /u01/app/oradata/CDB2 and /u01/app/oradata/CDB1/seed directories must exist.

Summary



Create a CDB by using the `CREATE DATABASE` command

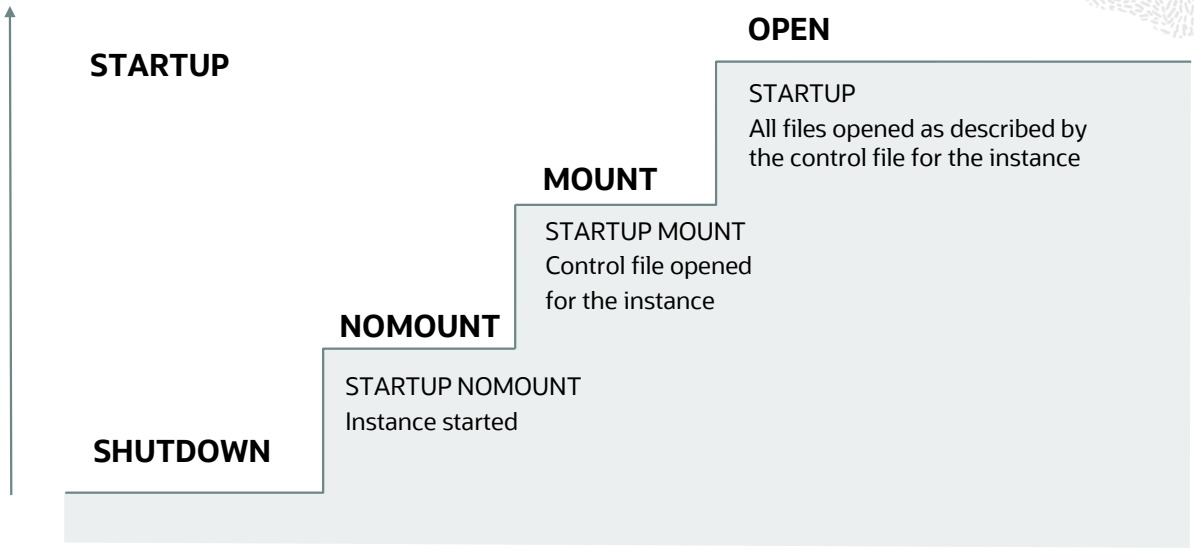
ORACLE
University

Starting Up and Shutting Down a Database Instance

Objectives

- Start up and shut down Oracle databases
- Open and close PDBs

Starting the Oracle Database Instance



Before users can connect to a database instance, a database administrator must start the database instance. The database instance and database go through stages as the database is made available for access by users. The database instance is started, the database is mounted, and then the database is opened, as shown in the slide.

You can use the `STARTUP` command in SQL*Plus with the options shown in the slide for each stage. The default option is `OPEN`.

`NOMOUNT`: During this stage, the Oracle software reads an initialization parameter file, starts background processes, allocates memory to the SGA, and opens the alert log and trace files. An instance is typically started only in `NOMOUNT` mode during database creation, during re-creation of control files, or in certain backup and recovery scenarios.

`MOUNT`: During this stage, the Oracle software associates the database (CDB) with the previously started database instance, opens and reads the control files that are specified in the initialization parameter file, and obtains the names and statuses of the data files and online redo log files. No checks, however, are performed to verify the existence of the data files and online redo log files at this time. Start in `MOUNT` mode to perform some maintenance operations, such as renaming data files and performing full database recoveries.

`OPEN`: The Oracle software opens the redo log files and data files according to the list registered in the control files. Start up in `OPEN` mode to enable users to connect to the database instance. PDBs are not, by default, started when you open the database.

Shutting Down an Oracle Database Instance

- Sometimes, you need to shut down the database instance (for example, to change a static parameter or patch the database server).
- Use the `SHUTDOWN` command to shut down the database instance in various modes: `ABORT`, `IMMEDIATE`, `NORMAL`, and `TRANSACTIONAL`.

	ABORT	IMMEDIATE	NORMAL	TRANSACTIONAL
Allows new connections	No	No	No	No
Waits until current sessions end	No	No	Yes	No
Waits until current transactions end	No	No	Yes	Yes
Forces a checkpoint and closes files	No	Yes	Yes	Yes

`ABORT` Mode: If the other shutdown modes don't work, you can use `ABORT` mode. `ABORT` mode performs the least amount of work before shutting down. Because this mode puts the database in an inconsistent state and requires recovery before startup, use it only when necessary. It's not advisable to back up the database in this state. It's typically used when no other form of shutdown works, when there are problems with starting the database instance, or when you need to shut down immediately because of an impending situation (such as notice of a power outage within seconds). `ABORT` mode is usually the fastest shutdown mode and `NORMAL` mode is the slowest. `NORMAL` and `TRANSACTIONAL` modes can take a long time depending on the number of sessions and transactions.

The following happens during a shutdown in `ABORT` mode, an instance failure, or a database instance startup in `FORCE` mode:

- Current SQL statements being processed by the Oracle server are immediately terminated.
- The Oracle server does not wait for users who are currently connected to the database to disconnect.
- Database and redo buffers are not written to disk.
- Uncommitted transactions are not rolled back.
- The instance is terminated without closing the files.
- The database is not closed or dismounted.
- The next startup requires instance recovery, which occurs automatically.

`IMMEDIATE` Mode: A shutdown in `IMMEDIATE` mode is the most typically used option.

- Current SQL statements being processed by the database instance are not completed.
- The database server does not wait for the users who are currently connected to the database instance to disconnect.
- The database server rolls back active transactions and disconnects all connected users.
- The database server closes and dismounts the database before shutting down the database instance.

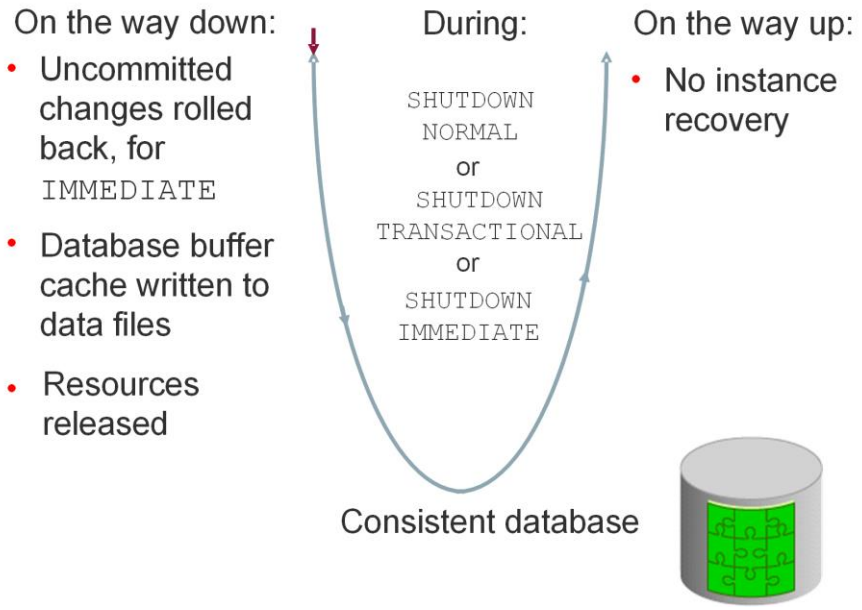
`NORMAL` Mode: `NORMAL` is the default shutdown mode if no mode is specified with the `SHUTDOWN` command.

- No new connections can be made.
- The Oracle server waits for all users to disconnect before completing the shutdown.
- Database and redo buffers are written to disk.
- Background processes are terminated, and the SGA is removed from memory.
- The Oracle server closes and dismounts the database before shutting down the instance.

`TRANSACTIONAL` Mode: A shutdown in `TRANSACTIONAL` mode prevents clients from losing data, including results from their current activity.

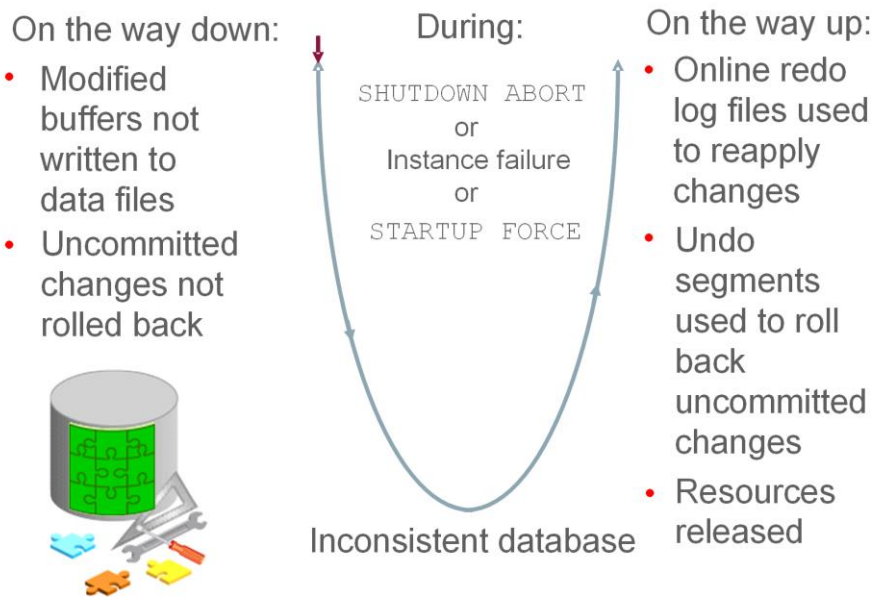
- No client can start a new transaction on this particular instance.
- A client is disconnected when the client ends the transaction that is in progress.
- When all transactions have been completed, a shutdown occurs immediately.

Comparing SHUTDOWN Modes



The diagram in this slide shows the IMMEDIATE, TRANSACTIONAL, and NORMAL shutdown modes. The diagram in the next slide shows ABORT shutdown mode (and instance failure or STARTUP FORCE mode). Notice that the database becomes inconsistent when you perform an ABORT shutdown, whereas it stays consistent during the other shutdown modes. Also note that you need to recover the database instance after you perform an ABORT shutdown, whereas with the other shutdown modes, you don't need to do so.

Comparing SHUTDOWN Modes



Notice that the database becomes inconsistent when you perform an ABORT shutdown, whereas it stays consistent during the other shutdown modes. Also, note that you need to recover the database instance after you perform an ABORT shutdown, whereas with the other shutdown modes, you don't need to do so.

Opening and Closing PDBs

- Open/close a PDB to open/close its data files.
- A PDB has four open modes:
  - READ WRITE (the PDB is fully started/opened)
  - READ ONLY
  - MIGRATE
  - MOUNTED (the PDB is shut down/closed)
- Use the ALTER PLUGGABLE DATABASE command or STARTUP and SHUTDOWN commands to open and close PDBs.
  - Example: SQL> ALTER PLUGGABLE DATABASE PDB1 OPEN;
- The ALTER PLUGGABLE DATABASE command lets you change from any open mode to another.
- To use the STARTUP command, the PDB must be in MOUNTED mode.

Open Modes

Starting up a PDB and opening a PDB mean the same thing, and you'll find both phrases used in documentation and online resources. When you open a PDB, the database server opens the data files for that PDB. Similar to a CDB, a PDB has four levels of being open, and these levels are referred to as open modes. The open modes are READ WRITE (the PDB is fully started/opened), READ ONLY, MIGRATE, and MOUNTED (the PDB is shut down/closed).

Commands to Open and Close PDBs

You can use the ALTER PLUGGABLE DATABASE command to open and close a PDB from either the root container or within the PDB itself. You can also use STARTUP and SHUTDOWN commands. The ALTER PLUGGABLE DATABASE command lets you change from any open mode to another for a PDB. To use the STARTUP command, the PDB must first be in MOUNTED mode. Either command requires you to be connected to the root container or PDB with one of the following system privileges: AS SYSBACKUP, AS SYSDBA, AS SYSDG, or AS SYSOPER.

Examples

In this example, PDB1 is started up (opened). Its open mode is changed from MOUNT to READ WRITE.

```
SQL> ALTER PLUGGABLE DATABASE PDB1 OPEN;
```

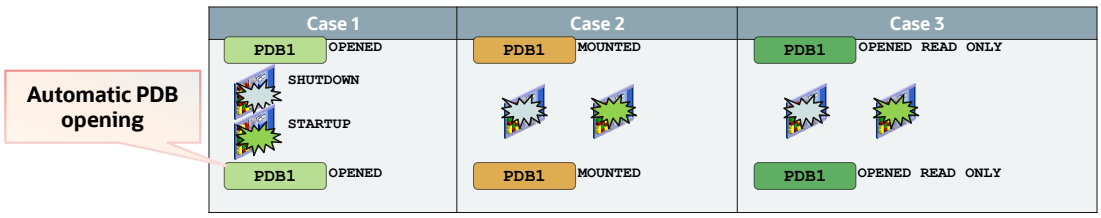
In this example, PDB1 is shut down (closed). Its open mode is changed to MOUNT.

```
SQL> ALTER PLUGGABLE DATABASE PDB1 CLOSE;
```

Configuring PDBs to Automatically Open

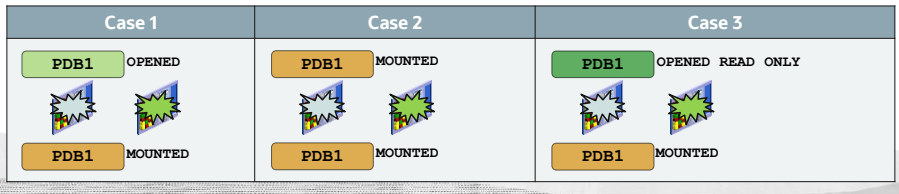
- Automatically keep the PDB's state after CDB `STARTUP`:

```
SQL> ALTER PLUGGABLE DATABASE pdb1 SAVE STATE;
```



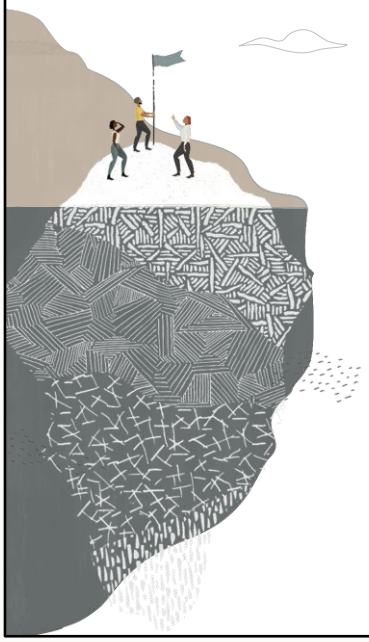
- Automatically discard the PDB's state after CDB `STARTUP`:

```
SQL> ALTER PLUGGABLE DATABASE pdb1 DISCARD STATE;
```



After restarting a CDB instance, the PDBs are by default kept in mounted mode. If you want the PDBs to automatically open whenever the CDB restarts, use the `SAVE STATE` clause of the `ALTER PLUGGABLE DATABASE` command to preserve a PDB's open mode across CDB restart. The `SAVE STATE` clause saves the last open state of the PDB. So, the PDB will open after the CDB restart only if the PDB was in the open state when the `SAVE STATE` clause was used to save the last state. To revert to the default behavior, use the `DISCARD STATE` clause.

Summary



- Start up and shut down Oracle databases
- Open and close PDBs

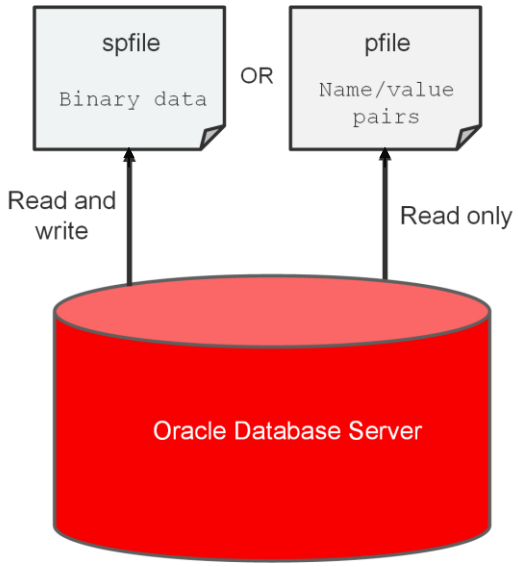
ORACLE
University

Managing Database Instances

Objectives

- Describe initialization parameter files and initialization parameters
- View and modify initialization parameters in SQL*Plus
- Work with the Automatic Diagnostic Repository (ADR)
- Query dynamic performance views

Working with Initialization Parameters



Initialization Parameter Files

When you start a database instance, it reads instance configuration parameters (initialization parameters) from an initialization parameter file (parameter file). On most platforms, parameter files are stored in the `$ORACLE_HOME/dbs` directory by default.

You can use one of the following types of parameter files to start your database instance, as illustrated in the slide:

- **Server parameter file (SPFILE):** An SPFILE is a binary file that is written to and read by the database server. You can't edit it manually. An SPFILE is preferred over a PFILE because you can change initialization parameters with `ALTER SYSTEM` commands in SQL*Plus, and the changes persist when you shut down and start up the database instance. It also provides a basis for self-tuning by Oracle Database. An SPFILE is automatically created for you by Database Configuration Assistant (DBCA) when you create a CDB. It resides on the server on which the Oracle instance is running. The default name of the SPFILE, which is automatically sought at startup, is `SPFILE<SID>.ora`.
- **Text initialization parameter file (PFILE):** A PFILE is a text file containing parameter values in name/value pairs, which the database server can read to start the database instance. Unlike an SPFILE, the database server cannot write to and alter a PFILE. Therefore, to change parameter values in a PFILE and make them persist during shutdown and startup, you must manually edit the PFILE in a text editor and restart the database instance to refresh the parameter values. Your installation includes a sample PFILE named `init.ora` in the default directory for parameter files. You can use this file as a starting point for a PFILE, or you can create a PFILE from the SPFILE. If you save your PFILE as `init<SID>.ora` in the default directory, the database server will automatically use it if an SPFILE is not available. If you save the PFILE under a different name, you'll need to specify it during startup.

Search Order for a Parameter File

The database server locates your parameter file by examining file names in the `$ORACLE_HOME/dbs` directory in the following order:

1. `SPFILE<SID>.ora`, where SID is the system ID and identifies the instance name (for example, `ORCL`)
2. `SPFILE.ora`
3. `init<SID>.ora` (PFILE)

Initialization Parameters

- Initialization parameters (parameters):
  - Set database limits
  - Set database-wide defaults
  - Specify files and directories
  - Affect performance
- Parameters can be of two types, basic or advanced.
  - Tune around 30 basic parameters to get reasonable database performance.
  - Example of a basic parameter: `SGA_TARGET`
  - Example of an advanced parameter: `DB_CACHE_SIZE`

Initialization parameters (parameters) set database limits, set databasewide defaults, specify files and directories, and affect performance. The parameter file must, at a minimum, specify the `DB_NAME` parameter. All other parameters have default values.

Types of Initialization Parameters

Parameters can be of two types: basic or advanced. In the majority of cases, you'll need to set and tune only the 30 or so basic parameters to get reasonable performance from the database. In rare situations, you'll need to modify one or more of the 300 or so advanced parameters to achieve optimal performance. An example of a basic parameter is `SGA_TARGET`, which specifies the total memory size of all SGA components. And example of an advanced parameter is `DB_CACHE_SIZE`, which specifies the size of the default buffer pool.

Initialization Parameters

- Derived parameters calculate their values from the values of other parameters.
  - Example: SESSIONS is derived from PROCESSES.
- Some parameter values or value ranges depend on the host operating system.
  - Example: DB_BLOCK_SIZE

Derived Parameters

Some parameters are derived, meaning their values are calculated from the values of other parameters. Normally, you shouldn't alter values for derived parameters. But if you do, the value that you specify overrides the calculated value. For example, the default value of the SESSIONS parameter is derived from the value of the PROCESSES parameter. If the value of PROCESSES changes, the default value of SESSIONS changes as well, unless you override it with a specified value.

Parameter Values That Depend on the OS

Some parameter values or value ranges depend on the host operating system. For example, the DB_FILE_MULTIBLOCK_READ_COUNT parameter specifies the maximum number of blocks that are read in one I/O operation during a sequential scan; this parameter is platform dependent. The size of those blocks, which is set by DB_BLOCK_SIZE, has a default value that depends on the operating system.

View

Review the information about the CONTROL_FILES parameter. *Oracle Database Reference* is a good source of information about parameters. In this example, you'll learn the parameter's data type (string), syntax (CONTROL_FILES = file name, [, file name]...), default value (operating system-dependent), whether its modifiable (no), whether you can modify its value in a PDB (no), its range of values (1 to 8 file names), whether it is a basic parameter (yes), and details for Oracle Real Application Clusters (multiple instances must have the same value).

Modifying Initialization Parameters

- Modify parameters to set capacity limits or improve performance.
  - Use Enterprise Manager or SQL*Plus (`ALTER SESSION` or `ALTER SYSTEM`).
- Query `V$PARAMETER` for an initialization parameter to learn whether you can make:
  - Session-level changes (`ISSES_MODIFIABLE` column)
  - System-level changes (`ISSYS_MODIFIABLE` column)
  - PDB-level changes (`ISPDB_MODIFIABLE` column)

Using `ALTER SESSION` or `ALTER SYSTEM` Commands

You modify parameters because you want to set capacity limits or improve performance. You can use the `ALTER SESSION` or `ALTER SYSTEM` commands in SQL*Plus to modify parameters. In your own environment, you'll likely only modify the basic initialization parameters to keep your database running with good performance.

Increasing the values of parameters may improve your system's performance, but increasing most parameters also increases the SGA size. A larger SGA can improve database performance up to a point. An SGA that is too large can degrade performance if it is swapped in and out of memory. You should set operating system parameters that control virtual memory working areas with the SGA size in mind. The operating system configuration can also limit the maximum size of the SGA.

Before modifying a parameter, you should query the `V$PARAMETER` view to learn about how you can modify a parameter.

- The `ISSES_MODIFIABLE` column value tells you whether you can change the parameter for your current session (`TRUE`) or not (`FALSE`) by using the `ALTER SESSION` command. You can change some parameters at the session level, but not all. Changes are applied to your current session immediately (dynamically) and expire when you end your session. Parameters with a value of `TRUE` are referred to as session-level parameters.

  Example: `SQL> ALTER SESSION SET NLS_DATE_FORMAT ='mon dd yyyy';`

Modifying Initialization Parameters

- Use the SCOPE clause with the ALTER SYSTEM command to tell the system where to update the system-level parameter:
  - MEMORY
  - SPFILE
  - BOTH
- Use the DEFERRED keyword to set or modify the value of the parameter for future sessions that connect to the database.

The ISSYS_MODIFIABLE column value tells you when a system-level change to the parameter, made by using the ALTER SYSTEM command, takes effect.

- IMMEDIATE means the change will take effect immediately and be applied to all current sessions.
- DEFFERED means the change will take effect in subsequent sessions.
- FALSE means the change will take effect in subsequent instances.
  - You can change all parameters at the system level by using the ALTER SYSTEM command, and the change is applied to all sessions.
  - Parameters with a value of FALSE are referred to as static parameters. For static parameters, you need to shut down and restart the database instance to implement the change. Also, the database instance must have been started with an SPFILE.
  - Example: `SQL> ALTER SYSTEM SET SEC_MAX_FAILED_LOGIN_ATTEMPTS=2 SCOPE=SPFILE;`

The ISPDB_MODIFIABLE column value tells you whether you can (TRUE) or can't (FALSE) modify the parameter inside a PDB. In a non-CDB, the value of this column is NULL.

Setting the Scope in the ALTER SYSTEM Command

Use the SCOPE clause with the ALTER SYSTEM command to tell the system where to update the system-level parameter. This location dictates how long the change will stay in effect. Scope also depends on whether you started the database instance by using a PFILE or an SPFILE. Scope can have the following values:

- MEMORY: This value tells the system to make the parameter change in memory only. The change will take effect immediately, but not persist in subsequent sessions. If you started the database instance by using a PFILE, then this is the only scope you can specify. This specification is not allowed for static parameters.
- SPFILE: This value tells the system to make the parameter change in the SPFILE only. The change will take effect immediately and persist after you restart the database instance. This is the only scope allowed for static parameters.
- BOTH: This value tells the system to make the parameter change in both memory and in the SPFILE. The change will take effect immediately and persist after you restart the database instance. If you started the database instance by using an SPFILE, then BOTH is the default.

Using the DEFERRED Keyword

The DEFFERED keyword tells the system to make the parameter change effective only for future sessions. You must specify DEFERRED in the ALTER SYSTEM command if the value of the ISSYS_MODIFIABLE column is DEFERRED. If the value of that column is IMMEDIATE, then the DEFERRED keyword is optional. If the value of that column is FALSE, then you cannot specify DEFERRED in the ALTER SYSTEM statement.

Viewing Initialization Parameters

- Ways to view initialization parameters in SQL*Plus:
  - Issue the SHOW PARAMETER command.
    - Example: Find out about all the parameters whose names contain the word “para.”

```
SQL> SHOW PARAMETER para
```

  - Query the following views:
    - V$PARAMETER
    - V$PARAMETER2
    - V$SPPARAMETER
    - V$SYSTEM_PARAMETER
    - V$SYSTEM_PARAMETER2

Issuing the SHOW PARAMETER Command

You can issue the SHOW PARAMETER command in SQL*Plus to view information about an initialization parameter (for example, view a parameter's data type and default value). For instance, the following SHOW PARAMETER command returns information about parameters whose names contain the word “para.”

```
SQL> SHOW PARAMETER para

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------------------
cell_offload_parameters              string
fast_start_parallel_rollback         string      LOW
parallel_adaptive_multi_user         boolean     TRUE
parallel_automatic_tuning            boolean     FALSE
```

Querying Views

You can also query the V$PARAMETER view in SQL*Plus to view information about an initialization parameter. For example, the following query against the V$PARAMETER view returns information about parameters whose names contain the word “pool.”

```
SQL> SELECT name, value FROM v$parameter WHERE name LIKE '%pool%';

NAME                         VALUE
---------------------------  ----------
shared_pool_size             0
large_pool_size              0
java_pool_size               0
streams_pool_size            0
shared_pool_reserved_size    15728640
…
9 rows selected.
```

Other views that contain parameter information include:

- `V$SPPARAMETER`: Displays information about the contents of the SPFILE. If you didn't use an SPFILE to start the database instance, each row of the view will contain `FALSE` in the `ISSPECIFIED` column.
- `V$PARAMETER2`: Displays information about the parameters that are currently in effect for the session, with each parameter value appearing as a row in the view. A new session inherits parameter values from the database instance-wide values displayed in the `V$SYSTEM_PARAMETER2` view.
- `V$SYSTEM_PARAMETER`: Displays information about the parameters that are currently in effect for the database instance
- `V$SYSTEM_PARAMETER2`: Displays information about the initialization parameters that are currently in effect for the instance, with each list parameter value appearing as a row in the view

Working with the Automatic Diagnostic Repository

- The Automatic Diagnostic Repository (ADR):
  - Is a file-based repository outside the database
  - Is a system-wide central tracing and logging repository
  - Stores database diagnostic data such as:
    - Traces
    - Alert log
    - Health monitor reports

Automatic Diagnostic Repository

The Automatic Diagnostic Repository (ADR) is a system-wide tracing and logging central repository for database diagnostic data such as traces, the alert log, health monitor reports, and more.

The ADR root directory is known as the ADR base. Its location is set by the `DIAGNOSTIC_DEST` initialization parameter (for example, `/u01/app/oracle`).

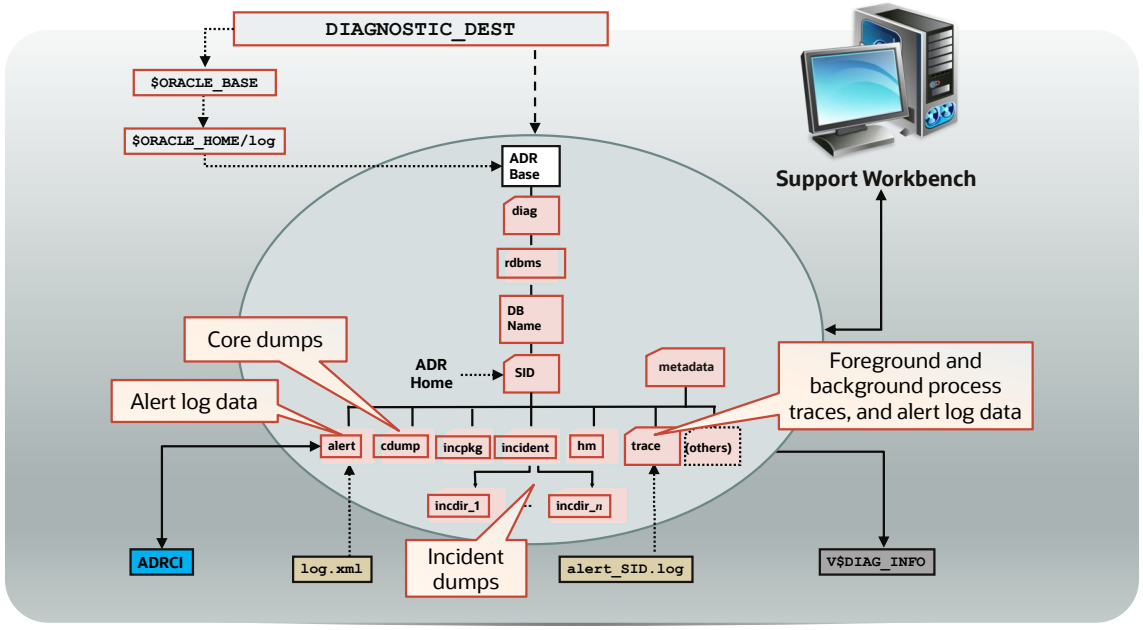
The location of an ADR home is given by the following path, which starts at the ADR base directory:

```
<ADR Base>/diag/product_type/db_id/instance_id
```

For example:

```
/u01/app/oracle/diag/rdbms/orcl/ORCL
```

Automatic Diagnostic Repository



ADR is a file-based repository for database diagnostic data such as traces, incident dumps and packages, the alert log, Health Monitor reports, core dumps, and more. It has a unified directory structure across multiple instances and multiple products—stored outside of any database. It is, therefore, available for problem diagnosis when the database is down.

The Oracle Database server, Automatic Storage Management (ASM), Cluster Ready Services (CRS), and other Oracle products or components store all diagnostic data in the ADR. Each instance of each product stores diagnostic data underneath its own ADR home directory. For example, in a Real Application Clusters environment with shared storage and ASM, each database instance and each ASM instance have a home directory within the ADR. ADR's unified directory structure, consistent diagnostic data formats across products and instances, and a unified set of tools enable customers and Oracle Support to correlate and analyze diagnostic data across multiple instances.

The ADR root directory is known as the ADR base. Its location is set by the `DIAGNOSTIC_DEST` initialization parameter. If this parameter is omitted or left null, the database sets `DIAGNOSTIC_DEST` upon startup as follows: If environment variable `ORACLE_BASE` is set, `DIAGNOSTIC_DEST` is set to `$ORACLE_BASE`. If environment variable `ORACLE_BASE` is not set, `DIAGNOSTIC_DEST` is set to `$ORACLE_HOME/log`.

Viewing the Alert Log

- The alert log file is a chronological log of messages about the database instance and database, such as:
  - Any nondefault initialization parameters used at startup
  - All internal errors (`ORA-600`), block corruption errors (`ORA-1578`), and deadlock errors (`ORA-60`) that occurred
  - Administrative operations, such as the SQL statements `CREATE`, `ALTER`, `DROP DATABASE`, and `TABLESPACE`, and the Enterprise Manager or SQL*Plus statements `STARTUP`, `SHUTDOWN`, `ARCHIVE LOG`, and `RECOVER`
  - Several messages and errors relating to the functions of shared server and dispatcher processes
  - Errors during the automatic refresh of a materialized view

Each database instance has an `alert_SID.log` file. The file is on the server with the database and is stored in `$ORACLE_BASE/diag/rdbms/<db_name>/<SID>/trace` by default if `$ORACLE_BASE` is set.

Oracle Database uses the alert log to keep a record of these events as an alternative to displaying the information on an operator's console. Many systems also display this information on the console. If an administrative operation is successful, a message is written in the alert log as "completed" along with a time stamp.

Enterprise Manager Cloud Control monitors the alert log file and notifies you of critical errors. You can also view the log to see noncritical error and information messages. Because the file can grow to an unmanageable size, you can periodically back up the alert file and delete the current alert file. When the database attempts to write to the alert file again, it creates a new one.

Note: There is an XML version of the alert log in the `$ORACLE_BASE/diag/rdbms/<db_name>/<SID>/alert` directory.

ADRCI is an Oracle command-line utility that enables you to investigate problems, view health check reports, and package and upload first-failure data to Oracle Support. You can also use the utility to view the names of the trace files in the Automatic Diagnostic Repository (ADR) and the alert log. ADRCI has a rich command set that you can use interactively or in scripts.

Viewing the Alert Log

- Query `V$DIAG_INFO` to find the location of the alert log.
  - The path to `alert_SID.log` corresponds to the Diag Trace entry.
  - The path to `log.xml` corresponds to the Diag Alert entry.
- You can view the alert log in a text editor or in ADRCI.

Using Trace Files

- Trace files contain:
  - Error information (contact Oracle Support Services if an internal error occurs)
  - Information that can provide guidance for tuning applications or an instance
- Each server and background process can write to an associated trace file.
- Trace file names for background processes are named after their processes.
  - Exception: Trace files generated by job queue processes
- Oracle Database includes an advanced fault diagnosability infrastructure for preventing, detecting, diagnosing, and resolving problems.

Trace Files

Each server and background process can write to an associated trace file. When a process detects an internal error, it dumps information about the error to its trace file. If an internal error occurs and information is written to a trace file, the administrator should contact Oracle Support Services.

All file names of trace files associated with a background process contain the name of the process that generated the trace file. The one exception to this is trace files that are generated by job queue processes (Jnnn).

Additional information in trace files can provide guidance for tuning applications or an instance. Background processes always write this information to a trace file when appropriate.

Oracle Database includes an advanced fault diagnosability infrastructure for preventing, detecting, diagnosing, and resolving problems. In particular, problems that are targeted include critical errors such as those caused by database code bugs, metadata corruption, and customer data corruption.

When a critical error occurs, an incident number is assigned to it; diagnostic data for the error (such as trace files) is immediately captured and tagged with this number. The data is then stored in the Automatic Diagnostic Repository (ADR)—a file-based repository outside the database—where it can later be retrieved by incident number and analyzed.

Using Trace Files

- When a critical error occurs:
  - An incident number is assigned to the error
  - Diagnostic data for the error (such as trace files) is immediately captured and tagged with the incident number
  - Data is stored in the ADR
- ADR files can be automatically purged by setting retention policy parameters.

Purging Mechanism

The purging mechanism allows you to specify a retention policy stating:

- How old ADR contents should be before they are automatically deleted
  - The long retention period is used for the relatively higher-value diagnostic data, such as incidents and alert log (default value is 365 days).
  - The short retention period is used for traces and core dumps (default value is 30 days).

  Older items are deleted first. The long retention period items are typically older than any of the items in the short retention period. So a mechanism is used in which the time periods are "scaled" so that roughly the same percentage of each gets deleted. Some components use these periods in slightly different ways. For instance, IPS, the packaging facility, uses the short retention period to determine when to purge packaging metadata and the staging directory contents. However, the age of the data is based on when the package was completed, not when it was originally created.
- Size-based retention to specify a target size for an ADR home

  When purging, the old data, determined by the time-based retention periods, is deleted first. If the size of the ADR home is still greater than the target size, diagnostics are automatically deleted until the target size is no longer exceeded.

Administering the DDL Log File

- Enable the capture of certain DDL statements to a DDL log file by setting `ENABLE_DDL_LOGGING` to `TRUE`
- The DDL log contains one log record for each DDL statement.
- Two DDL logs contain the same information:
  - XML DDL log: `log.xml` written to `$ORACLE_BASE/diag/rdbms/<dbname>/<SID>/log/ddl`
  - Text DDL: `ddl_<sid>.log` written to `$ORACLE_BASE/diag/rdbms/<dbname>/<SID>/log`

The DDL log is created only if the `ENABLE_DDL_LOGGING` initialization parameter is set to `TRUE`. When this parameter is set to `FALSE`, DDL statements are not included in any log. A subset of executed DDL statements is written to the DDL log.

There are two DDL logs that contain the same information. One is an XML file, and the other is a text file. The DDL log is stored in the `log/ddl` subdirectory of the ADR home.

Note: You must have a license for Oracle Database Lifecycle Management Pack to enable DDL logging.

Administering the DDL Log File

- Example:

```
$ more ddl_orcl.log
Thu Nov 15 08:35:47 2016
diag_adl:drop user app_user
```

Querying Dynamic Performance Views

- Dynamic performance views provide access to information about the changing states of instance memory structures:
  - Sessions, file states, and locks
  - Progress of jobs and tasks
  - Backup status, memory usage, and allocation
  - System and session parameters
  - SQL execution
  - Statistics and metrics

The Oracle Database server maintains a dynamic set of data about the operation and performance of the database instance. The dynamic performance views are based on virtual tables that are built from memory structures inside the database server. They are not conventional tables that reside in a database. This is the reason that some of them are available before a database is mounted or open.

Note: The `DICT` and `DICT_COLUMNS` views also contain the names of these dynamic performance views.

You can use dynamic performance views to answer questions such as the following:

1. For which SQL statements (and their associated numbers of executions) is the CPU time consumed greater than 200,000 microseconds?

   ```
   SQL> SELECT sql_text, executions FROM V$SQL  WHERE cpu_time > 200000;
   ```

2. What are the session IDs of those sessions that are currently holding a lock that is blocking another user, and how long have those locks been held?

   ```
   SQL> SELECT sid, ctime FROM v$lock WHERE block > 0;
   ```

Querying Dynamic Performance Views

- Dynamic performance views start with the prefix `v$`.
- Example query: Which current sessions have logged in from the EDXX9P1 computer on the last day?

```
SQL> SELECT * FROM V$SESSION
  2  WHERE machine = 'EDXX9P1'
  3  AND logon_time > SYSDATE - 1;
```

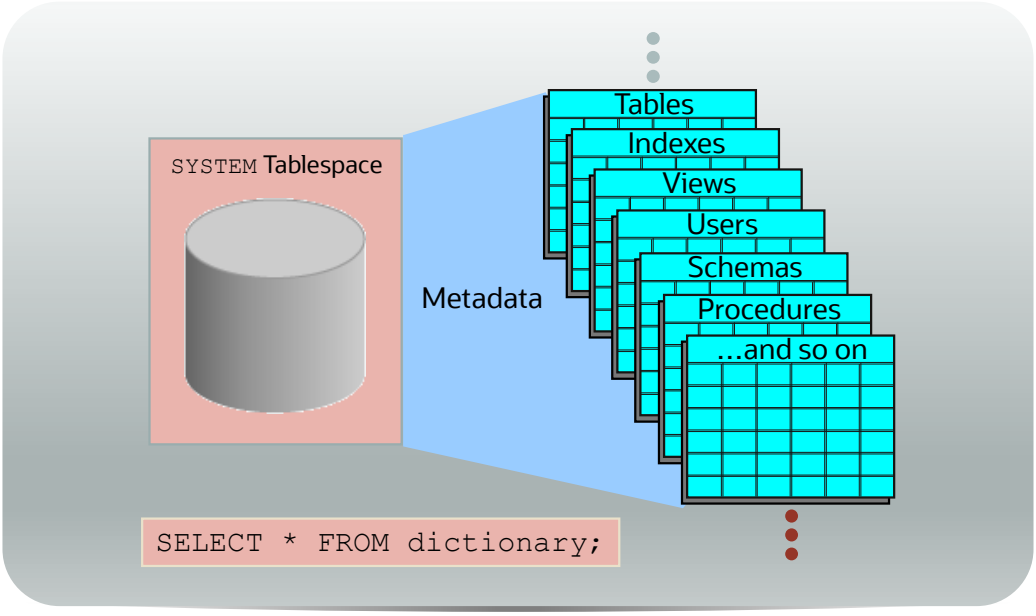
Considerations for Dynamic Performance Views

- These views are owned by the SYS user.
- Views provide information depending on the stage (NOMOUNT, MOUNT, or OPEN).
- You can query V$FIXED_TABLE to see all the view names.
- These views are often referred to as "v-dollar views."
- Read consistency is not guaranteed on these views because the data is dynamic.

Some dynamic views provide information that is not applicable to all states of an instance or database. For example, if an instance has just been started but no database is mounted, you can query V$BGPROCESS to see the list of background processes that are running. But querying V$DATAFILE to see the status of database data files would return no rows. The database must be mounted or opened for V$DATAFILE to provide meaningful information. It is when the database is mounted that the control file is read to obtain information about the data files associated with a database.

Some V$ views contain information that is similar to information in the corresponding DBA_ views. For example, V$DATAFILE is similar to DBA_DATA_FILES. Note also that V$ view names are generally singular and DBA_ view names are plural.

Data Dictionary: Overview



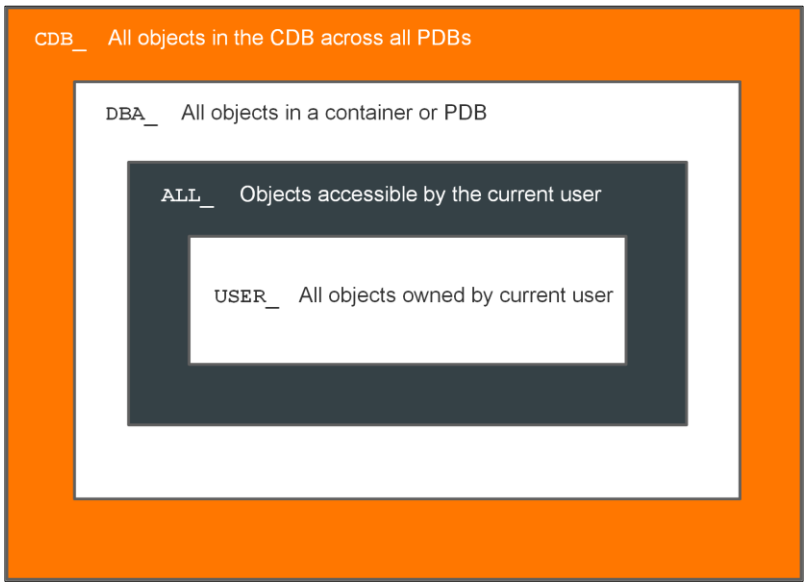
The Oracle data dictionary is the metadata of the database and contains the names and attributes of all objects in the database. The creation or modification of any object causes an update to the data dictionary that reflects those changes. This information is stored in the base tables that are maintained by the Oracle Database server, but you access these tables by using predefined views rather than by reading the tables directly.

The data dictionary:

- Is used by the Oracle Database server to find information about users, objects, constraints, and storage
- Is maintained by the Oracle Database server when object structures or definitions are modified
- Is available for use by any user to query information about the database
- Is owned by the `SYS` user
- Should never be modified directly by using SQL

Note: The `DICTIONARY` data dictionary view (or the `DICT` synonym for this) contains the names and descriptions of data dictionary tables and views. Use the `DICT_COLUMNS` view to see the view columns and their definitions. For complete definitions of each view, see the *Oracle Database Reference*. There are over 1000 views that reference hundreds of base tables.

Querying the Oracle Data Dictionary



CDB_, DBA_, ALL_, and USER_ Views

The view prefixes, as shown in the slide, indicate the data (and how much of that data) a given user can see.

- CDB_ views display metadata for all objects in a CDB across all PDBs.
- DBA_ views display metadata for all objects in a container or PDB.
- ALL_ views display metadata for objects that the current user is privileged to see, whether the user owns them or not. For example, if USER_A has been granted access to a table owned by USER_B, then USER_A sees that table listed in any ALL_ view dealing with table names.
- USER_ views display metadata for all objects owned by the current user, that is, objects that are present in the user's own schema.

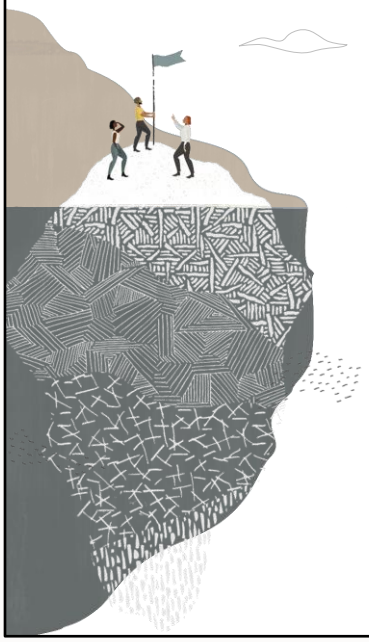
Only USER_ and ALL_ views are available to any user. The CDB_ and DBA_ views are restricted to DBA accounts.

Generally, each view set is a subset of the higher-privileged view set, row-wise and column-wise. Not all views in a given view set have a corresponding view in the other view sets. It depends on the nature of the information in the view. For example, there is a DBA_LOCK view, but no ALL_LOCK view, because only a DBA would have interest in data about locks. Be sure to choose the appropriate view set to meet the need that you have. If you have the privilege to access the DBA views, you still may want to query only the USER_ version of the view because the results show information on objects that you own, and you may not want other objects to be added to your result set.

The CDB_ and DBA_ views can be queried only by users with the SYSDBA or SELECT ANY DICTIONARY privilege, or SELECT_CATALOG_ROLE role, or by users with direct privileges granted to them.

When a user connected to the root queries a CDB_* view, the query results will depend on the CONTAINER_DATA attribute for the user. The CONTAINER_DATA clause of the SQL ALTER USER statement is used to set and modify the users' CONTAINER_DATA attribute. In a PDB, the CDB_* views only show objects visible through a corresponding DBA_* view.

Summary



- Describe initialization parameter files and initialization parameters
- View and modify initialization parameters in SQL*Plus
- Work with the Automatic Diagnostic Repository (ADR)
- Query dynamic performance views

ORACLE
University

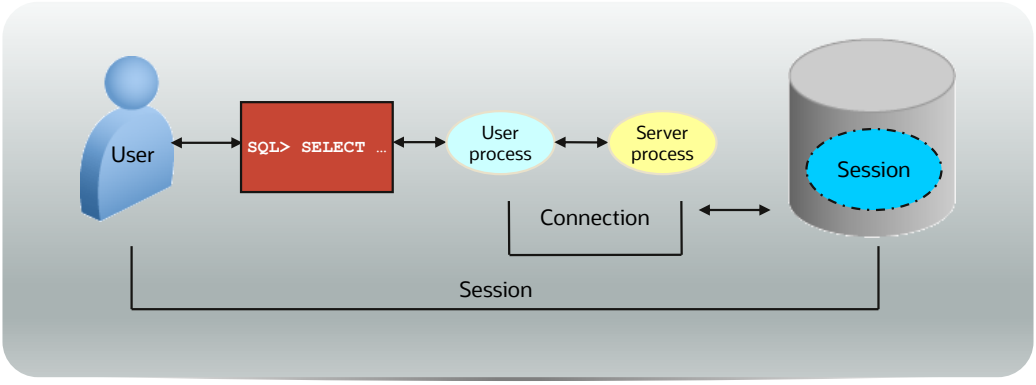
Oracle Net Services: Overview

Objectives

- List the components of Oracle Net Services
- Explain how listeners work
- Describe the tools that are used to administer Oracle Net Services
- Explain the difference between dedicated and shared server configurations

Connecting to the Database Instance

- **Connection:** Communication between a user process and an instance
- **Session:** Represents the state of a current user login to the database instance



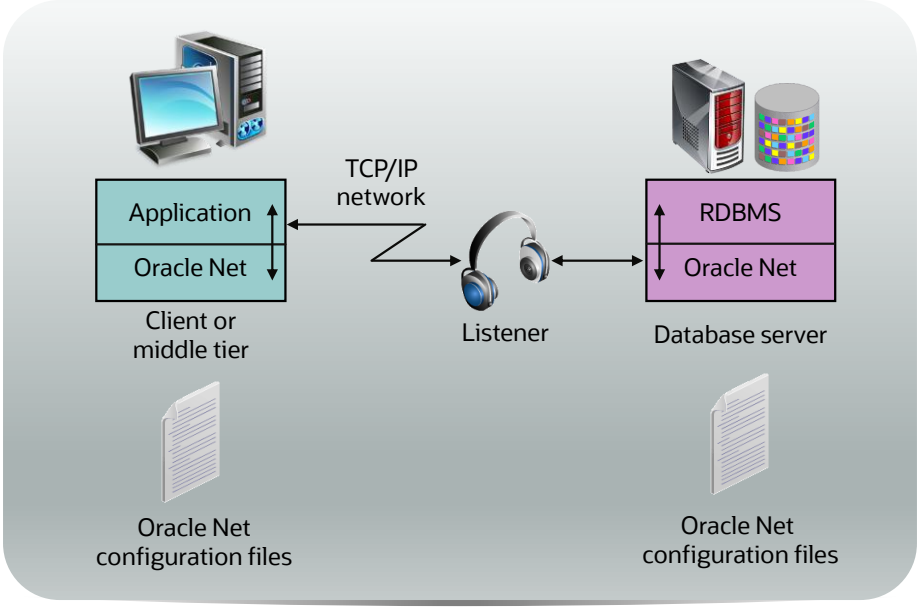
Connections and sessions are closely related to user processes but are very different in meaning.

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

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

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

Oracle Net Services: Overview



Oracle Net Services enables network connections from a client or middle-tier application to the Oracle server. After a network session is established, Oracle Net acts as the data courier for both the client application and the database server. It is responsible for establishing and maintaining the connection between the client application and database server, as well as exchanging messages between them. Oracle Net (or something that simulates Oracle Net, such as Java Database Connectivity) is located on each computer that needs to talk to the database server.

On the client computer, Oracle Net is a background component for application connections to the database.

On the database server, Oracle Net includes an active process called *Oracle Net Listener*, which is responsible for coordinating connections between the database and external applications.

The most common use of Oracle Net Services is to allow incoming database connections. You can configure additional net services to allow access to external code libraries (EXTPROC) and to connect the Oracle instance to non-Oracle data sources through Oracle Heterogeneous Services.

Defining Oracle Net Services Components

Component	Description	File
Listeners	A process that resides on the server whose responsibility is to listen for incoming client connection requests and manage traffic to the server	`listener.ora`
Naming methods	A resolution method used by a client application to resolve a connect identifier to a connect descriptor when attempting to connect to a database service	
Naming (net service name)	A simple name (connect identifier) for a service that resolves to a connect descriptor to identify the network location and identification of a service	`tnsnames.ora` (local configuration)
Profiles	A collection of parameters that specifies preferences for enabling and configuring Oracle Net features on the client or server	`sqlnet.ora`

The following Oracle Net Services components can be configured by using Enterprise Manager Cloud Control and Oracle Net Manager:

- Listener: Configuration of the listener includes specifying the listener name, protocol addresses it is accepting connection requests on, and services (database or nondatabase service) it is listening for.
- Naming methods
- Naming (net service name)
- Profiles

The Oracle Net Configuration Assistant configures the listener, naming methods, directory server usage, and a local `tnsnames.ora` file during the installation of Oracle Database software.

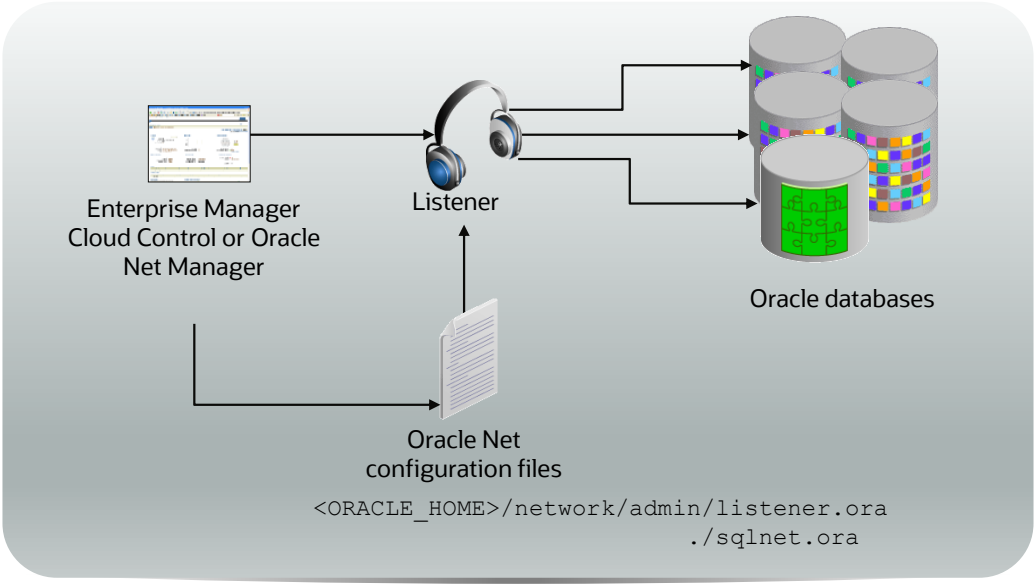
Tools for Configuring and Managing Oracle Net Services

- Enterprise Manager Cloud Control
- Oracle Net Manager
- Oracle Net Configuration Assistant
- Listener Control Utility

Use the following tools and applications to manage your Oracle Network configuration:

- **Enterprise Manager Cloud Control:** Provides an integrated environment for configuring and managing Oracle Net Services. Use Enterprise Manager to configure Oracle Net Services for any Oracle home across multiple file systems and administer listeners.
- **Oracle Net Manager:** Provides a graphical user interface (GUI) through which you can configure Oracle Net Services for an Oracle home on a local client or a server host
- **Oracle Net Configuration Assistant:** Launched by Oracle Universal Installer when you install the Oracle software. During a typical database installation, Oracle Net Configuration Assistant automatically configures a listener called LISTENER that has a TCP/IP listening protocol address for the database. If you perform a custom installation, Oracle Net Configuration Assistant prompts you to configure a listener name and protocol address of your choice.
- **Listener Control Utility:** Used to start, stop, and view the status of the listener process

Oracle Net Listener: Overview



Oracle Net Listener (or simply **the listener**) is the gateway to the Oracle instance for all nonlocal user connections. A single listener can service multiple database instances and thousands of client connections.

You can use Enterprise Manager Cloud Control or Oracle Net Manager to configure the listener and specify log file locations.

Advanced administrators can also configure Oracle Net Services by manually editing the configuration files, if necessary, with a standard operating system (OS) text editor such as `vi` or `gedit`.

The Default Listener

- During the creation of an Oracle database, the Oracle Net Configuration Assistant utility creates a local listener named `LISTENER`.
- `LISTENER` is automatically populated with available database services through a feature called dynamic service registration.
- `LISTENER` listens on the following TCP/IP protocol address:

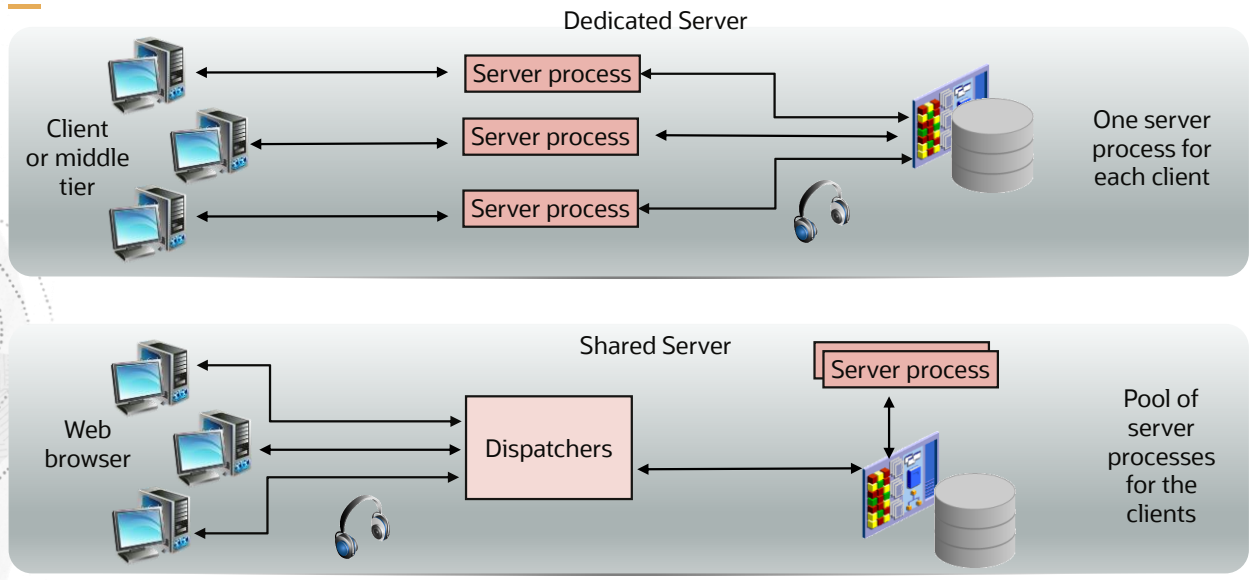
```
ADDRESS=(PROTOCOL=tcp)(HOST=host_name)(PORT=1521))
```

- Without any configuration, you can access your database instance immediately through `LISTENER`.
- If the listener name is `LISTENER` and it cannot be resolved, a protocol address of TCP/IP and a port number of 1521 is assumed.

Because the configuration parameters in the `listener.ora` file have default values, it is possible to start and use a listener with no configuration. This default listener has a name of LISTENER, supports no services on startup, and listens on the following TCP/IP protocol address:

```
(ADDRESS=(PROTOCOL=tcp)(HOST=host_name)(PORT=1521))
```

Comparing Dedicated and Shared Server Architecture



Dedicated Server Configuration

In a dedicated server configuration, as illustrated in the slide, one server process handles requests for a single client process. Each server process uses system resources, including CPU cycles and memory. In a heavily loaded system, the memory and CPU resources that are used by dedicated server processes can be prohibitive and negatively affect the system's scalability. If your system is being negatively affected by the resource demands of the dedicated server architecture, you have the following options:

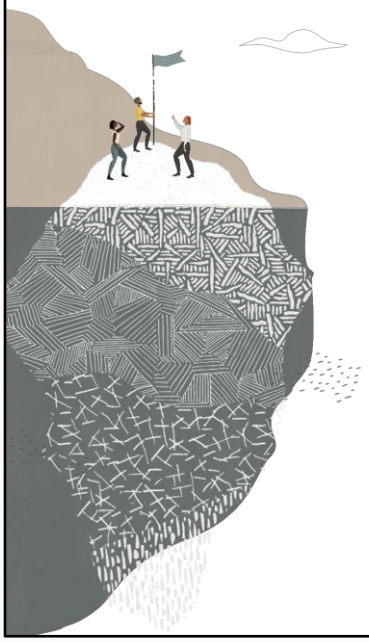
- Increase system resources by adding more memory and additional CPU capability
- Use the Oracle Shared Server Process architecture

Shared Server Configuration

A shared server configuration, as illustrated in the slide, enables multiple client processes to share a small number of server processes. Each service that participates in the shared server process architecture has at least one dispatcher process (and usually more). When a connection request arrives, the listener does not spawn a dedicated server process. Instead, the listener maintains a list of dispatchers that are available for each service name, along with the connection load (number of concurrent connections) for each dispatcher. Connection requests are routed to the lightest loaded dispatcher that is servicing a given service name. Users remain connected to the same dispatcher for the duration of a session.

Unlike dedicated server processes, a single dispatcher can manage hundreds of user connections. Dispatchers do not actually handle the work of user requests. Instead, they pass user requests to a common queue located in the shared pool portion of the SGA. Shared server processes take over most of the work of dedicated server processes, pulling requests from the queue and processing them until they are complete.

Summary



- List the components of Oracle Net Services
- Explain how listeners work
- Describe the tools that are used to administer Oracle Net Services
- Explain the difference between dedicated and shared server configurations

ORACLE
University

Configuring Naming Methods

Objectives

- Describe Oracle Net Services naming methods
- Configure local naming for database connections

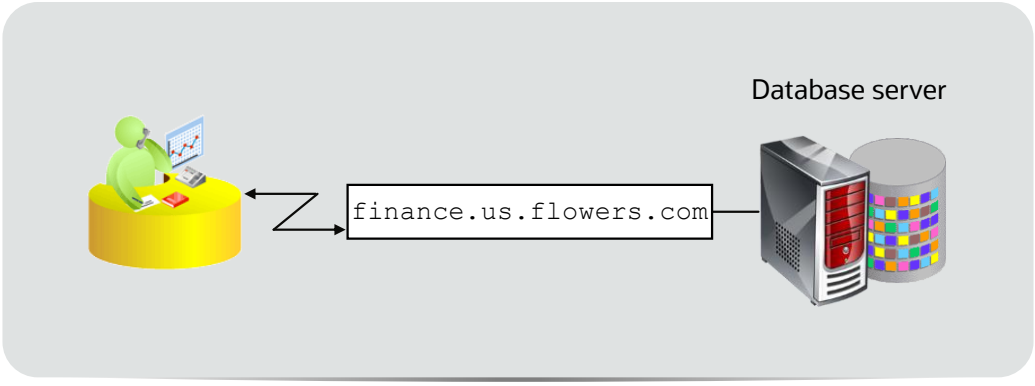
Establishing Oracle Network Connections

- To make a client or middle-tier connection, Oracle Net requires the client to know the:
  - Host where the listener is running
  - Port that the listener is monitoring
  - Protocol that the listener is using
  - Name of the service that the listener is handling

For an application to connect to a service through Oracle Net Listener, it must have information about that service, including the address or host where the listener resides, the protocol that the listener accepts, and the port that the listener monitors. After the listener is located, the final piece of information that the application needs is the name of the service to which it wants to connect.

Oracle Net names resolution is the process of determining this connection information.

Connecting to an Oracle Database Instance



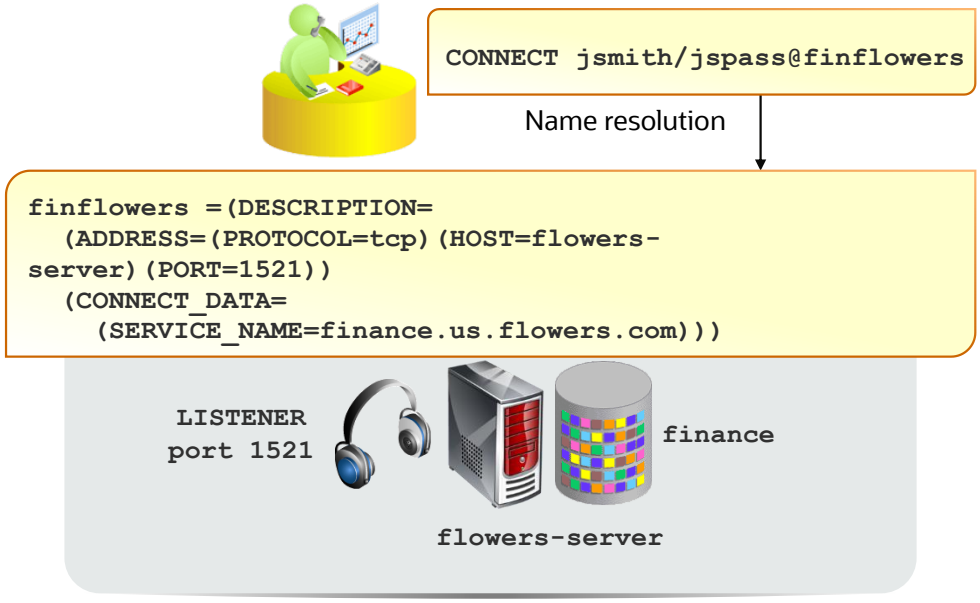
An Oracle database is represented to a client as a service. A database can have one or more services associated with it. Databases are identified by a **service name** that is specified by the `SERVICE_NAMES` parameter in the initialization parameter file. The service name defaults to the global database name, which is a name that comprises the database name (`DB_NAME` parameter value) and the domain name (`DB_DOMAIN` parameter value).

To connect to a database service, clients use a **connect descriptor** that provides the location of the database and the name of the database service. Clients can use the connect descriptor or a name that resolves to the connect descriptor (as discussed later in this lesson).

The following example shows a connect descriptor that enables clients to connect to a database service called `finance.us.flowers.com`.

```
(DESCRIPTION=
  (ADDRESS=(PROTOCOL=tcp)(HOST=flowers-server)(PORT=1521))
  (CONNECT_DATA=
    (SERVICE_NAME=finance.us.flowers.com)))
```

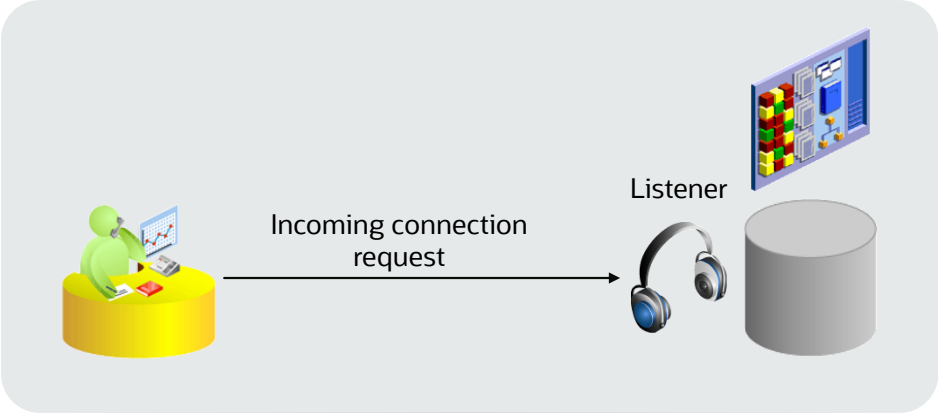
Name Resolution



Users initiate a connection request to the Oracle database instance by sending a **connect string**. A connect string includes a username and password, along with a **connect identifier**. A connect identifier can be the connect descriptor itself or a **name** that resolves to a connect descriptor. One of the most common connect identifiers is a **net service name**, which is a simple name for a service.

When a net service name is used, connection processing takes place by mapping the net service name to a connect descriptor. The mapping information can be stored in one or more repositories of information and is resolved by using a *naming method*.

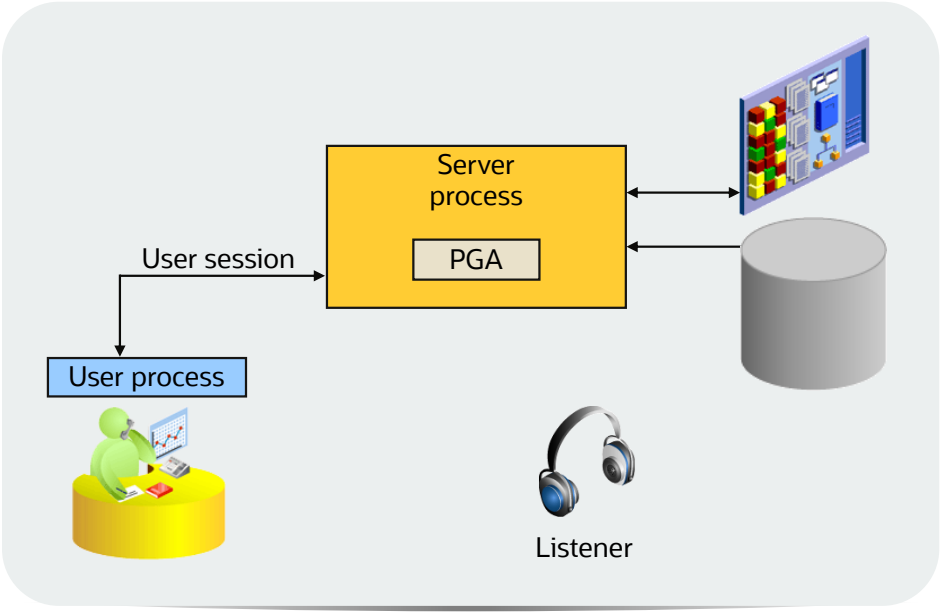
Establishing a Connection



After Oracle Net names resolution is complete, a connection request is passed from the user or middle-tier application (hereafter referred to as the *user process*) to the listener. The listener receives a `CONNECT` packet and checks whether that `CONNECT` packet is requesting a valid Oracle Net service name.

If the service name is not requested (as in the case of a `tnsping` request), the listener acknowledges the connect request and does nothing else. If an invalid service name is requested, the listener transmits an error code to the user process.

User Sessions



If the `CONNECT` packet requests a valid service name, the listener spawns a new process to deal with the connection. This new process is known as the **server process**. The listener connects to the process and passes the initialization information, including the address information for the user process. At this point, the listener no longer deals with the connection, and all work is passed to the server process.

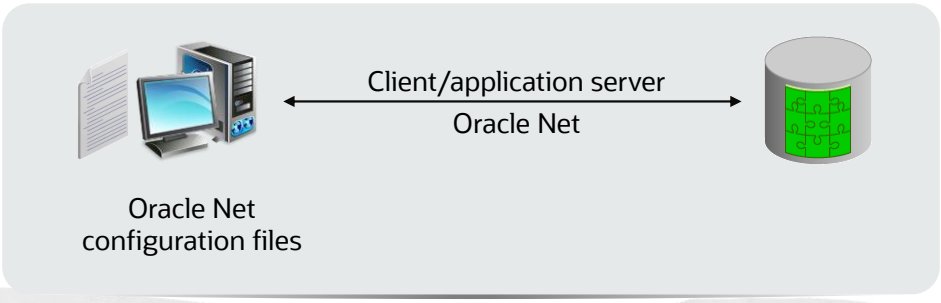
The server process checks the user's authentication credentials (usually a password), and if the credentials are valid, a user session is created.

Dedicated server process: With the session established, the server process now acts as the user's agent on the server. The server process is responsible for:

- Parsing and running any SQL statements issued through the application
- Checking the database buffer cache for data blocks required to perform SQL statements
- When required, reading necessary data blocks from data files on the disk into the database buffer cache portion of the System Global Area (SGA), if the blocks are not already present in the SGA
- Managing all sorting activity: The Sort Area is a memory area that is used to work with sorting; it is contained in a portion of memory that is associated with the Program Global Area (PGA).
- Returning results to the user process in such a way that the application can process the information
- Reading auditing options and reporting user processes to the audit destination

Naming Methods

- Oracle Net supports several methods for resolving connection information:
  - **Easy connect naming:** Uses a TCP/IP connect string
  - **Local naming:** Uses a local configuration file
  - **Directory naming:** Uses a centralized LDAP-compliant directory server



Oracle Net provides support for the following naming methods:

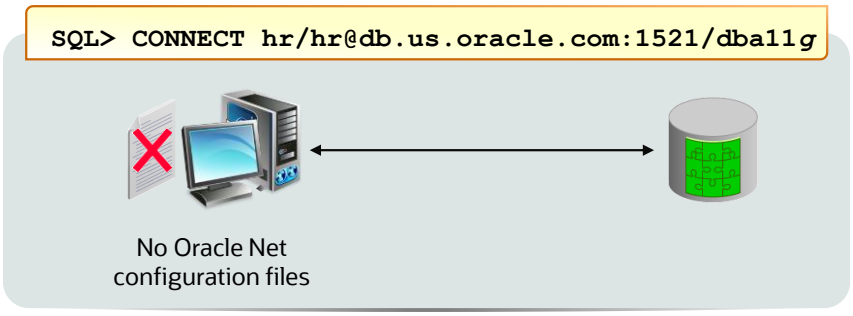
- **Easy connect naming:** The easy connect naming method enables clients to connect to an Oracle Database instance by using a TCP/IP connect string consisting of a host name, optional port, and service name as follows:

  ```
  CONNECT username/password@host[:port][/service_name]
  ```

  The easy connect naming method requires no configuration.
- **Local naming:** The local naming method stores connect descriptors (identified by their net service name) in a local configuration file named `tnsnames.ora` on the client.
- **Directory naming:** To access a database service, the directory naming method stores connect identifiers in a centralized directory server that is compliant with the Lightweight Directory Access Protocol (LDAP).
- **External naming:** The external naming method stores net service names in a supported non-Oracle naming service. Supported third-party services include:
  - Network Information Service (NIS) External Naming
  - Distributed Computing Environment (DCE) Cell Directory Services (CDS)

Easy Connect

- Is enabled by default
- Requires no client-side configuration
- Supports only TCP/IP (no SSL)
- Offers no support for advanced connection options such as:
  - Connect-time failover
  - Source routing
  - Load balancing



With Easy Connect, you supply all the information that is required for the Oracle Net connection as part of the connect string. Easy Connect connection strings take the following form:

```
<username>/<password>@<hostname>:<listener port>/<service name>
```

The listener port and service name are optional. If the listener port is not provided, Oracle Net assumes that the default port of 1521 is being used. If the service name is not provided, Oracle Net assumes that the database service name and host name provided in the connect string are identical.

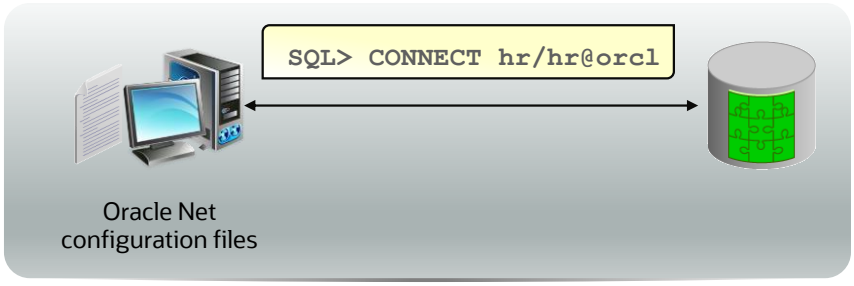
Assuming that the listener uses TCP to listen on port 1521 and the `SERVICE_NAMES=db` and `DB_DOMAIN=us.oracle.com` instance parameters, the connect string shown in the slide can be shortened:

```
SQL> connect hr/hr@db.us.oracle.com
```

Note: The `SERVICE_NAMES` initialization parameter can accept multiple comma-separated values. Only one of those values must be `db` for this scenario to work.

Local Naming

- Requires a client-side names-resolution file
- Supports all Oracle Net protocols
- Supports advanced connection options such as:
  - Connect-time failover
  - Source routing
  - Load balancing



With local naming, the user supplies an alias for the Oracle Net service. Oracle Net checks the alias against a local list of known services and, if it finds a match, converts the alias into host, protocol, port, and service name.

One advantage of local naming is that the database users need to remember only a short alias rather than the long connect string required by Easy Connect.

The local list of known services is stored in the following text configuration file:

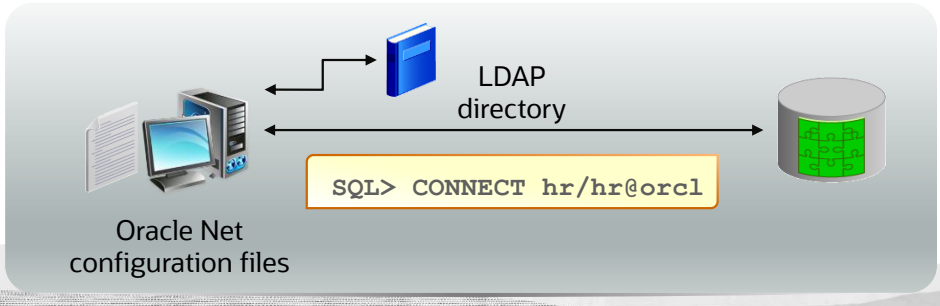
```
<oracle_home>/network/admin/tnsnames.ora
```

This is the default location of the `tnsnames.ora` file, but the file can be located elsewhere by using the `TNS_ADMIN` environment variable.

Local naming is appropriate for organizations in which Oracle Net service configurations do not change often.

Directory Naming

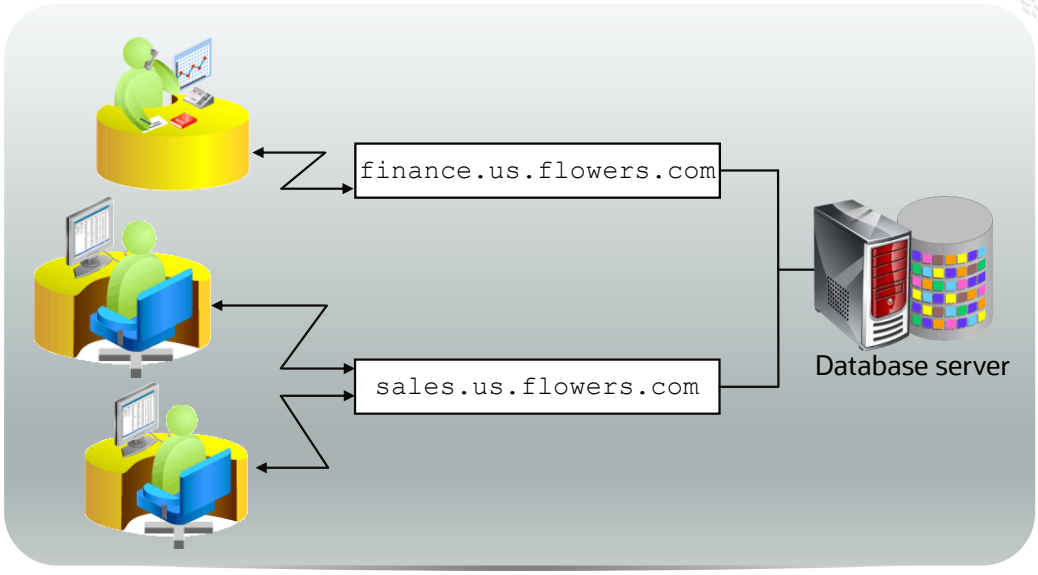
- Requires LDAP with Oracle Net names resolution information loaded:
  - Oracle Internet Directory
  - Microsoft Active Directory Services
- Supports all Oracle Net protocols
- Supports advanced connection options



With directory naming, the user supplies an alias for the Oracle Net service. Oracle Net checks the alias against an external list of known services and, if it finds a match, converts the alias into host, protocol, port, and service name. Like local naming, database users need to remember only a short alias.

Directory naming is appropriate for organizations in which Oracle Net service configurations change frequently.

Using Database Services to Manage Workloads



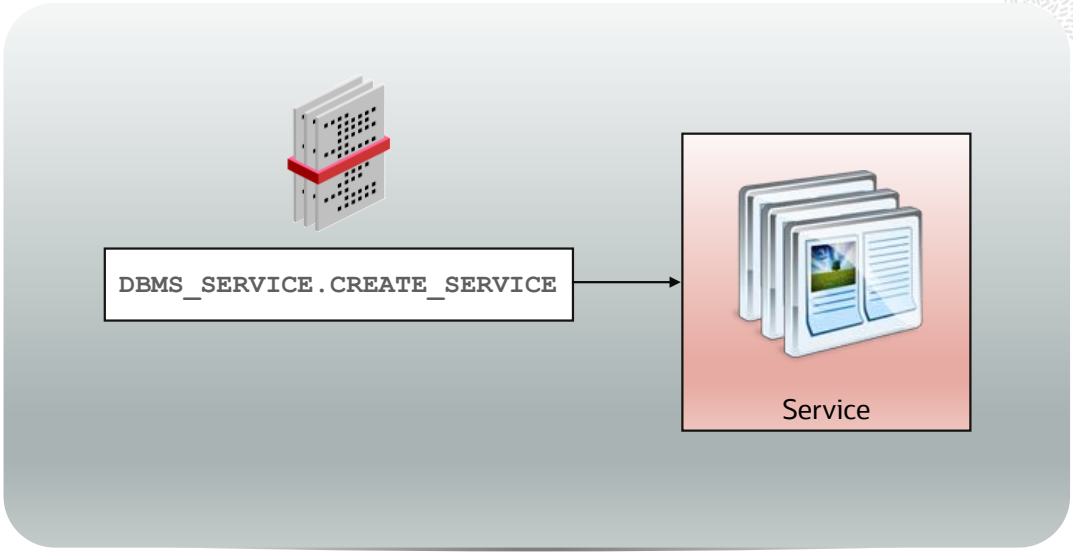
A database service is a named representation of a specific workload or application hosted by one or more database instances. Services enable you to group database workloads and route a particular work request to an appropriate instance.

Associating multiple services with one database enables the following functionality:

- A single database can be identified in different ways by different clients.
- System resources can be limited or reserved. This level of control enables better allocation of resources to clients requesting one of the services.

You can define services for PDBs. Each database service name must be unique in a CDB, and each database service name must be unique within the scope of all the CDBs whose instances are reached through a specific listener.

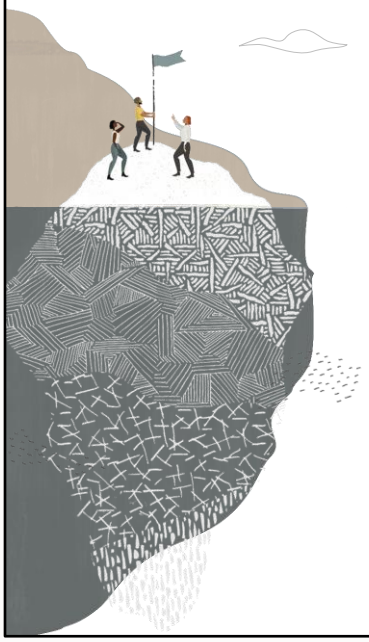
Creating Database Services



You can define a service by using the `DBMS_SERVICE` package and then use the net service name to assign applications to a service.

If your single-instance database is being managed by Oracle Restart or your Oracle RAC database is being managed by Oracle Clusterware, you should use the Server Control (SRVCTL) utility to create, modify, or remove the service.

Summary



- Describe Oracle Net Services naming methods
- Configure local naming for database connections

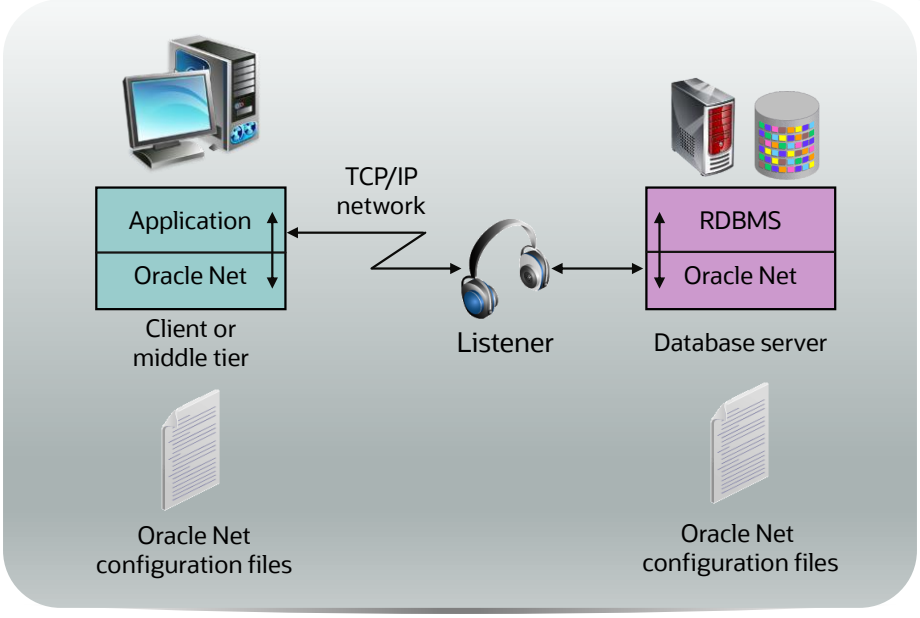
ORACLE
University

Configuring and Administering the Listener

Objectives

- Explain how listeners work
- Configure listeners for dynamic or static service registration

Review: Oracle Net Services Overview



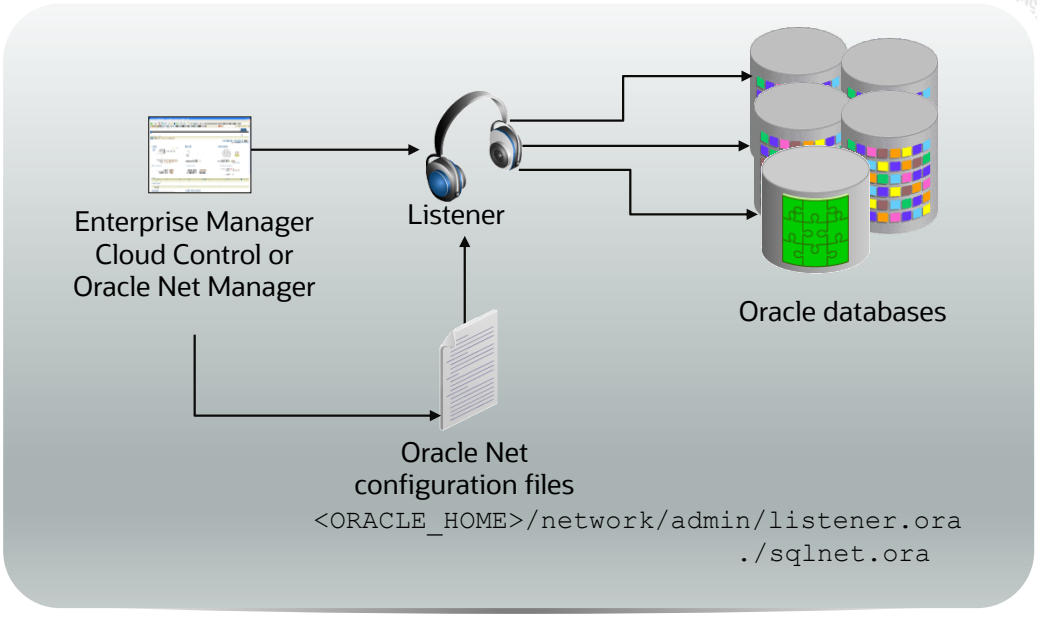
Oracle Net Services enables network connections from a client or middle-tier application to the Oracle server. After a network session is established, Oracle Net acts as the data courier for both the client application and the database server. It is responsible for establishing and maintaining the connection between the client application and database server, as well as exchanging messages between them. Oracle Net (or something that simulates Oracle Net, such as Java Database Connectivity) is located on each computer that needs to talk to the database server.

On the client computer, Oracle Net is a background component for application connections to the database.

On the database server, Oracle Net includes an active process called **Oracle Net Listener**, which is responsible for coordinating connections between the database and external applications.

The most common use of Oracle Net Services is to allow incoming database connections. You can configure additional net services to allow access to external code libraries (EXTPROC) and to connect the Oracle instance to non-Oracle data sources through Oracle Heterogeneous Services.

Oracle Net Listener: Overview



Oracle Net Listener (or simply **the listener**) is the gateway to the Oracle instance for all nonlocal user connections. A single listener can service multiple database instances and thousands of client connections.

You can use Enterprise Manager Cloud Control or Oracle Net Manager to configure the listener and specify log file locations.

Advanced administrators can also configure Oracle Net Services by manually editing the configuration files, if necessary, with a standard operating system (OS) text editor such as `vi` or `gedit`.

The Default Listener

- During the creation of an Oracle database, the Oracle Net Configuration Assistant creates a local listener named `LISTENER`.
- `LISTENER` is automatically populated with available database services through a feature called *dynamic service registration*.
- `LISTENER` listens on the following TCP/IP protocol address:

```
ADDRESS=(PROTOCOL=tcp)(HOST=host_name)(PORT=1521))
```

- Without any configuration, you can access your database instance immediately through `LISTENER`.

The default listener has a name of `LISTENER`, manages no services on startup, and listens on the following TCP/IP protocol address:

```
(ADDRESS=(PROTOCOL=tcp)(HOST=host_name)(PORT=1521))
```

Configuring Dynamic Service Registration

- By default, an Oracle database instance is configured to use dynamic service registration (service registration), which allows the Oracle database instance to identify its available services to listeners automatically.
- The LREG process polls the listeners to see if they are running and, if so, registers database service information to them.
- Dynamic service registration registers, by default, all PDB services to the same listener. If you stop that listener, you stop access to all the PDB services.
- General steps to configure dynamic service registration:
  - Make sure that the `INSTANCE_NAME` , `LOCAL_LISTENER`, `REMOTE_LISTENER`, and `SERVICE_NAMES` initialization parameters are properly configured.
  - Configure protocol addresses (end points) in the server-side `tnsnames.ora` file.
- Use the `ALTER SYSTEM REGISTER` command to initiate service registration immediately after the listener is started.

Benefits of Dynamic Service Registration

Service registration offers the following benefits:

- Connect-time failover: Because the listener always monitors the state of the instances, service registration facilitates automatic failover of a client connect request to a different instance if one instance is down.
- Connection load balancing: Service registration enables the listener to forward client connect requests to the least-loaded instance and dispatcher or dedicated server. Service registration balances the load across the service handlers and nodes.
- High availability for Oracle Real Application Clusters and Oracle Data Guard

The Role of the LREG Process

The Listener Registration (LREG) process polls the listeners to see if they are running and, if so, registers the following database service information to them:

- Database instance name
- Database service names available on the database instance (for example, `ORCL.example.com` and `PDB1.example.com`)
- Current and maximum load for the database instance
- Service handlers (dispatchers and dedicated servers) available to the database instance

LREG registers with the listeners after the database instance mounts the database and every 60 seconds afterward. You can use the `ALTER SYSTEM REGISTER` command to initiate service registration immediately after the listener is started.

How to Configure Dynamic Service Registration

The LREG process learns of the available listeners through the LOCAL_LISTENER and REMOTE_LISTENER parameters. These parameters specify listener alias names for local listeners and remote listeners. Both parameters can have multiple values. These aliases resolve to protocol addresses (end points) in the server-side tnsnames.ora file.

Note: Clients can also have a tnsnames.ora file, which you'll learn about in a later lesson.

Through dynamic service registration, the LREG process is then able to pass on information about the available database services to all listeners on those end points.

For example, assume LOCAL_LISTENER = LISTENER_HOST1 and REMOTE_LISTENER = LISTENER_HOST2. In the tnsnames.ora file, the LISTENER_HOST1 and LISTENER_HOST2 aliases are resolved to two different end points on two different machines. Notice that the CONNECT_DATA section is not included.

```
LISTENER_HOST1 =

  (ADDRESS = (PROTOCOL = TCP)(HOST = host1.example.com)(PORT = 1521))

LISTENER_HOST2 =

  (ADDRESS = (PROTOCOL = TCP)(HOST = host2.example.com)(PORT = 1521))
```

For dynamic service registration to work properly, make sure that the INSTANCE_NAME, LOCAL_LISTENER, REMOTE_LISTENER, and SERVICE_NAMES parameters are configured properly. By default, the installer populates the SERVICE_NAME parameter with the global database name (for example, ORCL.example.com), which provides one database service name that users can use to access the database instance. You can specify multiple service names for the database instance, however, if you want to distinguish among different uses of the same database. Oracle Database Resource Manager lets you view information about the user activity for each service name.

Dynamic service registration does not use the listener.ora file.

Configuring Static Service Registration

- Static service registration is a method for configuring listeners to obtain their service information manually.
  - You can create a listener for a particular PDB.
  - Static service registration might be required for some services, such as external procedures and heterogeneous services (for non-Oracle systems).
- With static registration, the listener has no knowledge of whether its database services exist. It only knows that it supports them. The Listener Control utility shows the services status as `UNKNOWN`.
- General steps to configure static service registration:
  1. In `listener.ora`, define a listener and its protocol addresses.
  2. In `listener.ora`, also create a `SID_LIST_<listener name>` section that lists the database services for the listener.

Advantages of Static Service Registration

The following are some advantages of using static service registration:

- Static service registration enables you to create a listener for a particular PDB.
- Sometimes you may need the database instance up and running without anyone being able to log in. As soon as it is started up, dynamic service registration will automatically start registering all the database services to the listener, making the database instance available to users.
- There is also a difference in error messages returned between a static listener (which can point to a database service that is down) and a dynamic listener entry (which shows nonexistence) when the database instance is shut down. The first case knows about the database service's existence and gives you an error message with useful information. The second case has no information and can't distinguish between a typo you may have made in the service name and whether it actually even exists.

Required Use of Static Service Registration

Static service registration is required for the following:

- Use of external procedure calls
- Use of Oracle Heterogeneous Services
- Use of Oracle Data Guard
- Remote database startup from a tool other than Oracle Enterprise Manager Cloud Control

How to Configure Static Service Registration

To create a static listener, you configure the `listener.ora` file. In that file, you define two sections. First, define the listener and its protocol addresses. Then, create a `SID_LIST_<listener name>` section that lists the database services for the listener. For each service, include the following parameters:

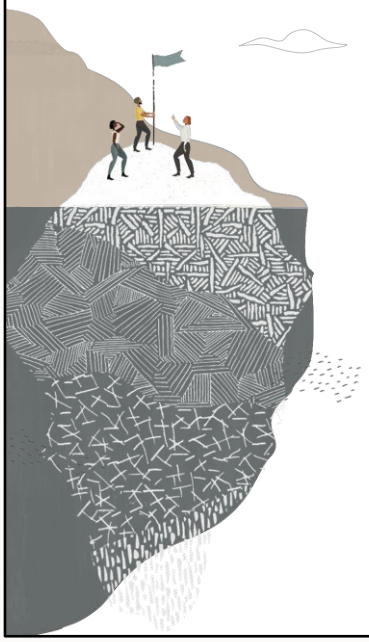
- `GLOBAL_DBNAME`: The PDB's service name (for example, `PDB1.example.com`)
- `ORACLE_HOME`: The Oracle home directory
- `SID_NAME`: The name of your database instance (for example, `ORCL`)

By default, the `listener.ora` file is stored in the `$ORACLE_HOME/network/admin` directory on the database instance machine.

```
LISTENER_SALESPDBS =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = host1.example.com)(PORT = 1561))
      ))


SID_LIST_LISTENER_SALESPDBS =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = PDB1.example.com)
      (SID_NAME = ORCL)
      (ORACLE_HOME = /u01/app/oracle/product/23.3.0/dbhome_1)
     )
    (SID_DESC =
      (GLOBAL_DBNAME = PDB2.example.com)
      (SID_NAME = ORCL)
      (ORACLE_HOME = /u01/app/oracle/product/23.3.0/dbhome_1)
    )
  )
```

Summary



- Explain how listeners work
- Configure listeners for dynamic or static service registration

ORACLE
University

Configuring a Shared Server Architecture

Objectives

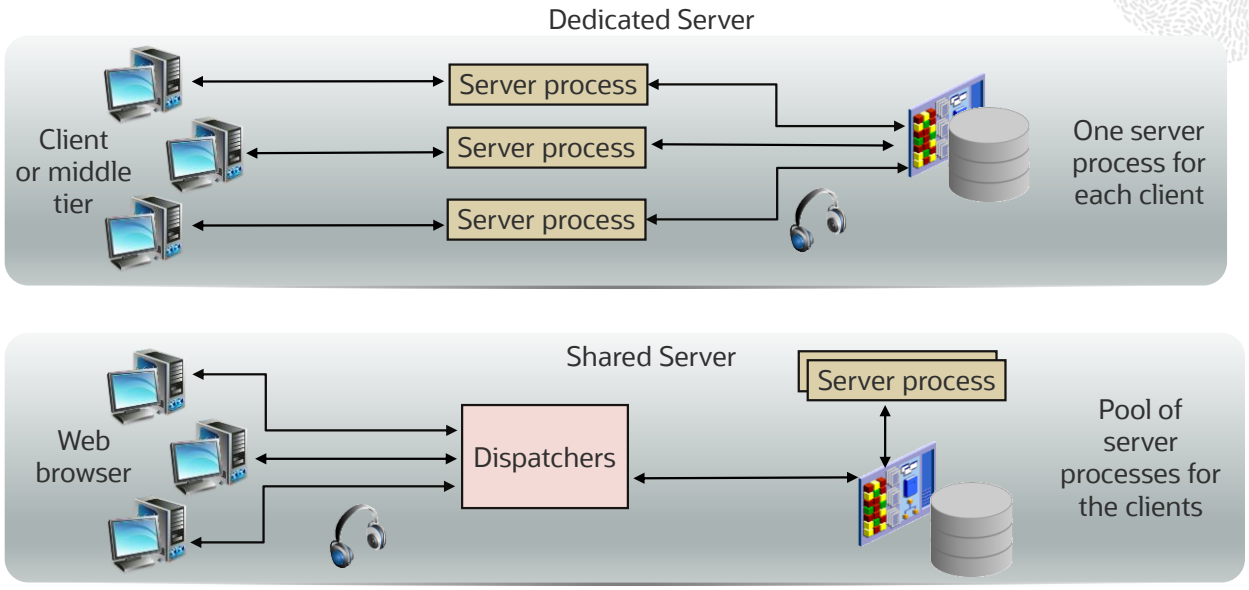
- Explain the difference between dedicated and shared server configurations
- Enable shared server
- Control shared server operations

Shared Server Architecture: Overview

- When should you consider configuring the shared server architecture?
  - System is overloaded.
  - System has limited memory.
- How does the shared server architecture address these issues?
  - Client processes share server processes.
  - Few processes are required to support the client load.

When client load causes a strain on memory and other system resources, you may be able to alleviate load issues by implementing the shared server architecture. This architecture enables client processes to share server processes, so the number of supported users is increased. A small pool of server processes can serve a large number of clients. This is useful when a system is overloaded or has limited memory.

Comparing Dedicated and Shared Server Architecture: Review



Dedicated Server Configuration

In a dedicated server configuration, as illustrated in the slide, one server process handles requests for a single client process. Each server process uses system resources, including CPU cycles and memory. In a heavily loaded system, the memory and CPU resources that are used by dedicated server processes can be prohibitive and negatively affect the system's scalability. If your system is being negatively affected by the resource demands of the dedicated server architecture, you have the following options:

- Increase system resources by adding more memory and additional CPU capability
- Use the Oracle Shared Server Process architecture

Shared Server Configuration

A shared server configuration, as illustrated in the slide, enables multiple client processes to share a small number of server processes. Each service that participates in the shared server process architecture has at least one dispatcher process (and usually more). When a connection request arrives, the listener does not spawn a dedicated server process. Instead, the listener maintains a list of dispatchers that are available for each service name, along with the connection load (number of concurrent connections) for each dispatcher. Connection requests are routed to the lightest loaded dispatcher that is servicing a given service name. Users remain connected to the same dispatcher for the duration of a session.

Unlike dedicated server processes, a single dispatcher can manage hundreds of user connections. Dispatchers do not actually handle the work of user requests. Instead, they pass user requests to a common queue located in the shared pool portion of the SGA. Shared server processes take over most of the work of dedicated server processes, pulling requests from the queue and processing them until they are complete.

Enabling Shared Server

- SHARED_SERVERS specifies the minimum number of shared servers that will be created when the instance is started.
- Enable shared server by setting the SHARED_SERVERS initialization parameter to a value greater than 0 in the initialization parameter file or by using the ALTER SYSTEM command.
- If SHARED_SERVERS is not in the initialization parameter file and DISPATCHERS is set to 1 or greater, then shared server is enabled by default.

The SHARED_SERVERS initialization parameter specifies the minimum number of shared servers that you want created when the instance is started. After instance startup, the Oracle Database server can dynamically adjust the number of shared servers based on how busy existing shared servers are and the length of the request queue.

If SHARED_SERVERS is not included in the initialization parameter file at database startup, but DISPATCHERS is included and it specifies at least one dispatcher, then shared server is enabled. In this case, the default for SHARED_SERVERS is 1.

Controlling Shared Server Operations

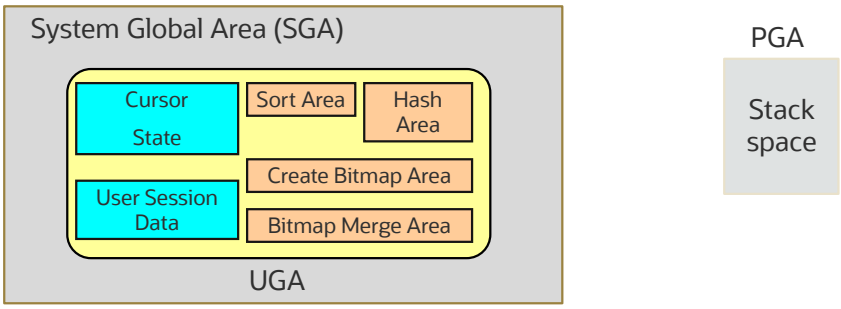
Initialization Parameter	Used To
DISPATCHERS	Configure dispatcher processes
SHARED_SERVERS	Specify the initial number of shared servers to start and the minimum number of shared servers to keep
MAX_SHARED_SERVERS	Specify the maximum number of shared servers that can run simultaneously
SHARED_SERVER_SESSIONS	Specify the total number of shared server user sessions that can run simultaneously
CIRCUITS	Set a maximum limit on the number of virtual circuits that can be created in shared memory

The following initialization parameters are used to control shared server operations:

- DISPATCHERS: At least one dispatcher process is required for a shared server configuration. If you do not specify a dispatcher, but you enable shared server by setting SHARED_SERVER to a nonzero value, then by default the Oracle Database server creates one dispatcher for the TCP protocol.
- SHARED_SERVERS: You can set the value in the initialization parameter file or use the ALTER SYSTEM command to configure the initial number of shared servers. After instance startup, the Oracle Database server can dynamically adjust the number of shared servers based on how busy existing shared servers are and the length of the request queue.
- MAX_SHARED_SERVERS: Use this parameter to limit the maximum number of shared servers that can be automatically created by PMON. The primary reason to limit the number of shared servers is to reserve resources for other processes.
- SHARED_SERVER_SESSIONS: Use this parameter to limit the number of concurrent shared server user sessions. It provides a way for you to reserve database sessions for dedicated servers.
- CIRCUITS: You can use this parameter to protect shared memory by limiting the number of virtual circuits that can be created in shared memory.

SGA and PGA Usage

- Oracle Shared Server: User session data is held in the SGA.
- Be sure to consider shared server memory requirements when sizing the SGA.



Because a user session may have requests processed by multiple shared server processes, most of the memory structures that are usually stored in the PGA must be in a shared memory location (by default, in the shared pool). However, if the large pool is configured or if `SGA_TARGET` is set for automatic memory management, these memory structures are stored in the large pool portion of the SGA.

The contents of the SGA and PGA are different in a shared server configuration from those in a dedicated server configuration. In a shared server configuration:

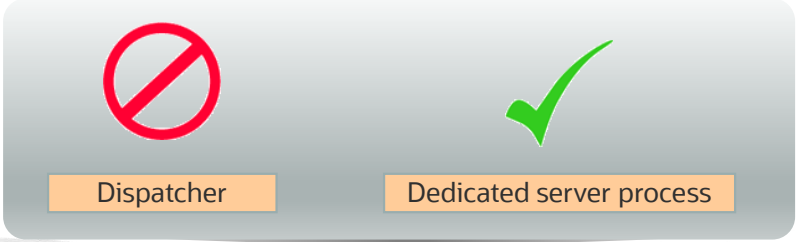
- Text and parsed forms of all SQL statements are stored in the SGA
- The cursor state contains runtime memory values for the SQL statement, such as rows retrieved
- User-session data includes security and resource usage information
- The stack space contains local variables for the process

The change in the SGA and PGA is transparent to the user; however, if you are supporting multiple users, you need to increase the `LARGE_POOL_SIZE` initialization parameter. Each shared server process must access the data spaces of all sessions so that any server can handle requests from any session. Space is allocated in the SGA for each session's data space. You limit the amount of space that a session can allocate by setting the `PRIVATE_SGA` resource.

Note: You do not need to set `LARGE_POOL_SIZE` when automatic memory management or automatic shared memory management are configured.

Shared Server Configuration Considerations

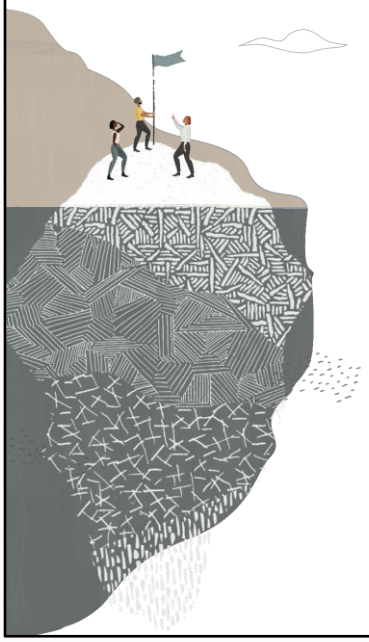
- Certain types of database work must not be performed using shared servers:
  - Database administration
  - Backup and recovery operations
  - Batch processing and bulk-load operations
  - Data warehouse operations



A dedicated server process is good for long-running queries and administrative tasks and is faster than a shared server process in that there is always a server process ready to do work. However, an idle process or too many dedicated processes can result in an inefficient use of resources. Using shared server mode on the database server eliminates the need for a dedicated server process for each user connection, requires less memory for each user connection, and enables a larger number of users on a system with constrained memory. Dedicated server processes and shared server processes are enabled at the same time. Oracle XML DB (XDB) requires shared server processes, and the Oracle database is already configured to use them. You will need to modify the initialization parameters for other users to use shared server processes.

The Oracle Shared Server architecture is an efficient process and memory use model, but it is not appropriate for all connections. Because of the common request queue and the fact that many users may share a dispatcher response queue, shared servers do not perform well with operations that must deal with large sets of data, such as warehouse queries or batch processing. Backup and recovery sessions that use Oracle Recovery Manager also deal with very large data sets and must use dedicated connections. Many administration tasks must not (and cannot) be performed by using shared server connections. These include starting up and shutting down the instance, creating tablespaces and data files, maintaining indexes and tables, analyzing statistics, and many other tasks that are commonly performed by the DBA. All DBA sessions must choose dedicated servers.

Summary



- Explain the difference between dedicated and shared server configurations
- Enable shared server
- Control shared server operations

Practice Overview

- This practice covers the following topics:
- Configuring Shared Server Mode
- Configuring Clients to Use a Shared Server

ORACLE
University

Creating PDBs from Seed

Objectives

- Create a new PDB from the PDB seed

Provisioning New Pluggable Databases

- Create a new PDB from the PDB seed.
- Plug an unplugged PDB into the same CDB or into another CDB.
- Plug a non-CDB in a CDB as a PDB.
- Clone a PDB from another PDB (local or remote CDB, hot or cold).
- Relocate a PDB from a CDB into another CDB.
- Proxy a PDB from another PDB.

There are different methods to provision new PDBs in a CDB.

- Create a new PDB from the PDB seed, `PDB$SEED`, for a new application implementation as an example. This type of PDB creation is nearly instantaneous.
- Plug an unplugged PDB into another CDB or into the same CDB. For example, you have to upgrade a PDB to the latest Oracle version, but you do not want to upgrade all PDBs. Instead of upgrading a CDB from one release to another, you can unplug a PDB from one Oracle Database release and then plug it into a newly created CDB at a higher release. If you mistakenly unplugged a PDB, you can replug it into the original CDB.
- Plug non-CDBs into a CDB as PDBs, as part of a migration strategy. It is also a good way to consolidate several non-CDBs into a CDB.
- Clone a PDB from another PDB of the same CDB. For example, you want to test an application patch. You first clone your production application in a cloned PDB and patch the cloned PDB to test.
- Relocate a PDB into another CDB to better allocate resources.
- Proxy a PDB. A proxy PDB provides fully functional access to another PDB in a remote CDB. This feature enables you to build location-transparent applications that can aggregate data from multiple sources that are in the same data center or distributed across data centers.

Tools

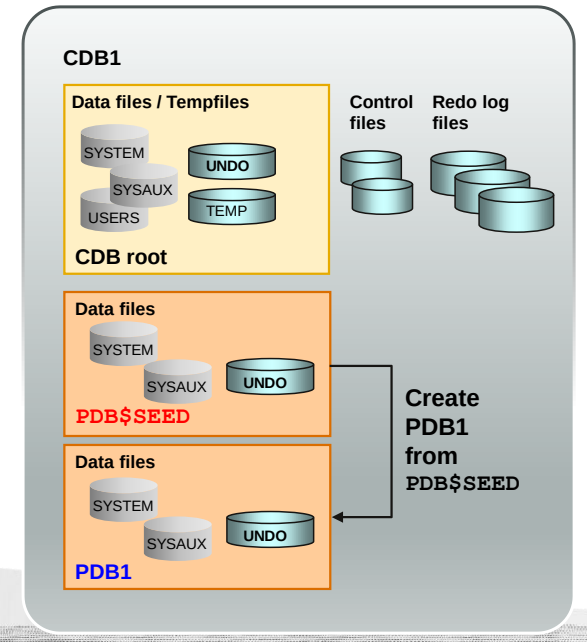
- To provision new PDBs, you can use:
  - SQL*Plus
  - SQL Developer
  - Enterprise Manager Cloud Control
  - Enterprise Manager Database Express
  - Database Configuration Assistant (DBCA)
    - Clone from PDB seed
    - Clone from an existing PDB
    - Plug an unplugged PDB

There are different tools to provision new PDBs in a CDB:

- SQL*Plus
- SQL Developer
- Enterprise Manager Cloud Control
- Enterprise Manager Database Express

To create a new PDB from the PDB seed or from an existing PDB or by plugging an unplugged PDB method, you can also use Database Configuration Assistant (DBCA).

Creating a New PDB from PDB$SEED



- Copies the data files from PDB$SEED data files
- Creates tablespaces:
  - SYSTEM
  - SYSAUX
  - UNDO
- Creates a full catalog including metadata pointing to Oracle-supplied objects
- Creates common users:
  - SYS
  - SYSTEM
- Creates a local user (PDBA), granted local PDB_DBA role
- Creates a new default service

The creation of a new PDB from the PDB seed is nearly instantaneous. The operation copies the data files from the READ ONLY seed PDB to the target directory defined in the CREATE PLUGGABLE DATABASE statement.

It creates tablespaces such as SYSTEM, to store a full catalog including metadata pointing to Oracle-supplied objects, SYSAUX for local auxiliary data, and UNDO for local undo segments.

It creates default schemas and common users that exist in the PDB seed, SYS that continues to have all super user privileges, and SYSTEM that can administer the PDB.

It creates a local user (the PDBA), granted a local PDB_DBA role. Until the PDB SYS user grants privileges to the local PDB_DBA role, the new PDBA cannot perform any operation other than connecting to the PDB.

A new default service is also created for the PDB.

Using the FILE_NAME_CONVERT Clause

Create a new PDB from the seed using **FILE_NAME_CONVERT**:

1. Connect to the CDB root as a common user with the CREATE PLUGGABLE DATABASE system privilege:

```
SQL> CREATE PLUGGABLE DATABASE pdb1
     ADMIN USER admin1 IDENTIFIED BY p1 ROLES=(CONNECT)
     FILE_NAME_CONVERT = ('PDB$SEEDdir', 'PDB1dir');
```

2. Use views to verify:

```
SQL> CONNECT / AS SYSDBA
SQL> SELECT * FROM cdb_pdbs;
SQL> SELECT * FROM cdb_tablespaces;
SQL> SELECT * FROM cdb_data_files;
SQL> ALTER PLUGGABLE DATABASE pdb1 OPEN RESTRICTED;
SQL> CONNECT sys@pdb1 AS SYSDBA
SQL> CONNECT admin1@pdb1
```

Note: The STATUS of the PDB is NEW.

The steps to create a new PDB from the PDB seed are the following:

1. Connect to the CDB root as a common user with the CREATE PLUGGABLE DATABASE system privilege and execute the CREATE PLUGGABLE DATABASE statement as shown in the slide. The ADMIN USER clause defines the PDBA user created in the new PDB with the CONNECT and PDB_DBA roles (empty role). The clause FILE_NAME_CONVERT designates first the source directory of the PDB seed data files and second the destination directory for the new PDB data files.
2. When the statement completes, use views to verify that the PDB is correctly created. The CDB_PDBS view displays the list of the PDBs and the CDB_TABLESPACES view displays the list of the tablespaces of the new PDB (SYSTEM, SYSAUX, UNDO). The CDB_PDBS view shows the STATUS of the new PDB: it is NEW. The PDB has never been opened. It must be opened in READ WRITE or RESTRICTED mode for Oracle to perform processing that is needed to complete the integration of the PDB into the CDB and mark it NORMAL. An error will be returned if an attempt is made to open the PDB read only.
3. Still connected to the CDB root, open the PDB. Then, try to connect to the new PDB under common user, SYS, which always exists in any PDB, or the user defined in the ADMIN USER clause, admin1.

Using OMF or the PDB_FILE_NAME_CONVERT Parameter

- Use OMF: **DB_CREATE_FILE_DEST =** '/u01/app/oradata/CDB1/**pdb1**'

  Or

- Use the initialization parameter: **PDB_FILE_NAME_CONVERT =** '/u01/app/oradata/CDB1/seed','/u01/app/oradata/CDB1/**pdb1**'

```
SQL> CREATE PLUGGABLE DATABASE pdb1
     ADMIN USER pdb1_admin IDENTIFIED BY p1 ROLES=(CONNECT);
```

Or

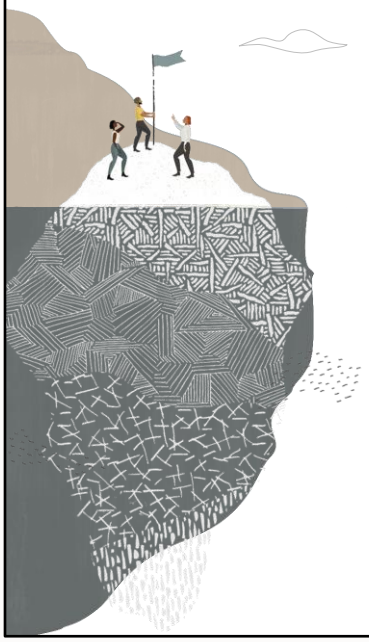
- Use the clause in the CREATE PLUGGABLE DATABASE command: **CREATE_FILE_DEST =** '/u01/app/oradata/CDB1/**pdb1**'

If you use Oracle Managed Files (OMF) or PDB_FILE_NAME_CONVERT:

1. Connect to the CDB root as SYS
2. With OMF, set the DB_CREATE_FILE_DEST initialization parameter to a target directory for the data files of the new PDB
3. Without OMF, set the PDB_FILE_NAME_CONVERT initialization parameter to both the source directory of the PDB seed data files and the target directory for the new PDB data files. In the example shown, the /u01/app/oradata/CDB1/pdb1 directory must exist.
4. Then, use the cdb_pdbs view to verify that the new PDB and its tablespaces exist:

```
SQL> SELECT * FROM cdb_pdbs;
SQL> SELECT * FROM cdb_tablespaces;
SQL> SELECT * FROM cdb_data_files;
```

Summary



Create a new PDB from the PDB seed

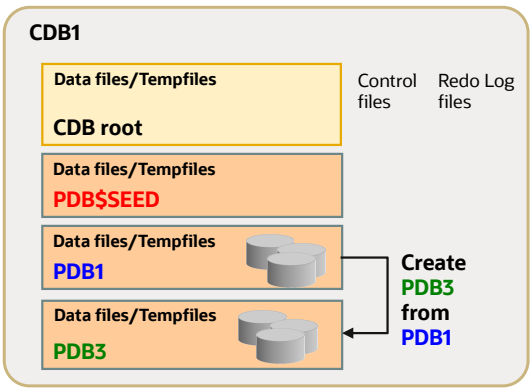
ORACLE
University

Using Other Techniques to Create PDBs

Objectives

- Clone a regular PDB
- Unplug and plug or clone a non-CDB
- Unplug and plug a regular PDB
- Perform hot cloning
- Perform near-zero downtime PDB relocation
- Create and use a proxy PDB

Cloning Regular PDBs



PDB3 owns:

- SYSTEM, SYSAUX, UNDO tablespaces
- Full catalog
- SYS, SYSTEM common users
- Same local administrator name
- New service name

1. Define the data file location:
   - Set DB_CREATE_FILE_DEST= 'PDB3dir'
   - Set PDB_FILE_NAME_CONVERT='PDB1dir', 'PDB3dir'
   - Use the CREATE_FILE_DEST= 'PDB3dir' clause
2. Connect to the CDB root to close **PDB1**.
3. Clone **PDB3** from **PDB1**.

```
SQL> CREATE PLUGGABLE DATABASE pdb3 FROM pdb1
          CREATE_FILE_DEST = 'PDB3dir';
```

4. Open **PDB3** in read write mode.

```
SQL> ALTER PLUGGABLE DATABASE pdb3 OPEN;
```

Note: Clone metadata only by using NO DATA.

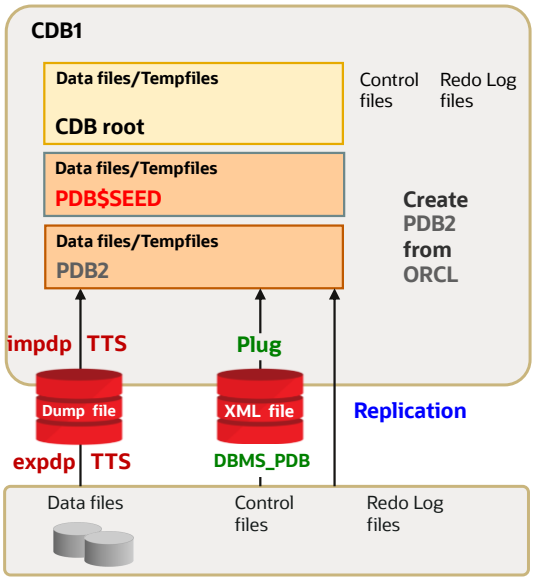
This technique copies a source PDB from a CDB and plugs the copy into the same CDB. The source PDB is in the local CDB.

The steps to clone a PDB within the same CDB are the following:

1. Define the location for the data files of the new PDB:
   - Set initialization parameters: DB_CREATE_FILE_DEST= 'PDB3dir' if you are using Oracle Managed Files (OMF) or PDB_FILE_NAME_CONVERT= 'PDB1dir', 'PDB3dir' for non-OMF.
   - Use the CREATE_FILE_DEST clause during the CREATE PLUGGABLE DATABASE statement (OMF).
   - Use the FILE_NAME_CONVERT=('pdb1dir',' pdb3dir') clause to define the directory of the source files to copy from PDB1 and the target directory for the new files of PDB3 (non-OMF).
2. Connect to the CDB root as a common user with the CREATE PLUGGABLE DATABASE privilege.
3. Use the command CREATE PLUGGABLE DATABASE to clone the pdb3 from pdb1.
4. Then open the new pdb3 with the command ALTER PLUGGABLE DATABASE OPEN.

   Note: NO DATA allows PDB metadata cloning, which is an option that can be used to clone an empty PDB and later import data for testing.

Migrating Data from a Non-CDB into a CDB



Possible methods:

- Data Pump (TTS or TDB or full export/import)
- Plugging (XML file definition with `DBMS_PDB`)
- Cloning
- Replication

Entities created in the new PDB:

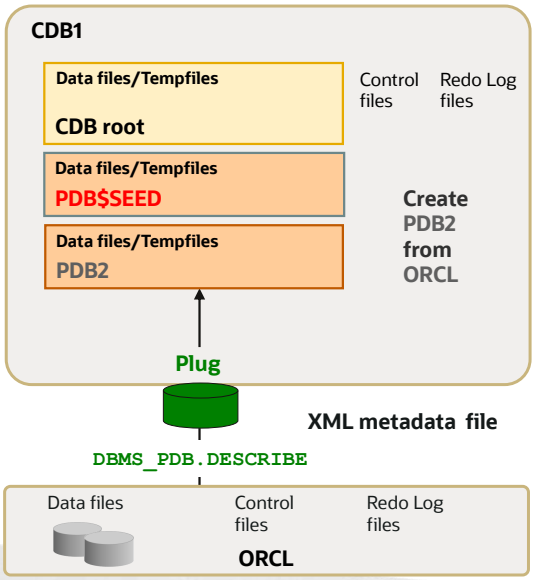
- Tablespaces: `SYSTEM`, `SYSAUX`, `UNDO`
- A full catalog
- Common users: `SYS`, `SYSTEM`
- A local administrator (PDBA)
- A new default service

There are different possible methods to migrate data from a non-CDB database into a CDB.

Whichever method is used, you have to get the non-CDB into a transactionally consistent state and open it in restricted mode.

- It is appropriate to use Oracle Data Pump when:
  - Both source and target databases are different endian
  - The source character set is not equal to the target character set and is not a binary subset of the target
  - Use either transportable tablespace (TTS) or full conventional export/import or full transportable database (TDB), provided that in the last one, any user-defined object resides in a single user-defined tablespace. Data Pump full transportable database does not support movement of XDB or AWR repositories. Only user-generated XML schemas are moved.
- In other cases, using the `DBMS_PDB` package is the easiest option. The `DBMS_PDB` package constructs an XML file describing the non-CDB data files to plug the non-CDB into the CDB as a PDB. It is also a good way to quickly consolidate several non-CDBs into a CDB.
- Cloning non-CDBs in a CDB is a good way to keep the non-CDB and therefore have the opportunity to compare the performance between the new PDB and the original non-CDB or at least wait until you consider that the PDB can work appropriately.

Plugging a Non-CDB into CDB Using DBMS_PDB



1. Open **ORCL** in READ ONLY mode.

```
SQL> EXEC DBMS_PDB.DESCRIBE ('/tmp/ORCL.xml')
```

2. Connect to the target CDB root as a common user with the CREATE PLUGGABLE DATABASE privilege.
3. Plug in the unplugged **ORCL** as **PDB2.**

```
SQL> CREATE PLUGGABLE DATABASE PDB2
             USING '/tmp/ORCL.xml';
```

4. Run the noncdb_to_pdb.sql script in **PDB2**.

```
SQL> CONNECT sys@PDB2 AS SYSDBA
SQL> @$ORACLE_HOME/rdbms/admin/noncdb_to_pdb
```

5. Open PDB2.

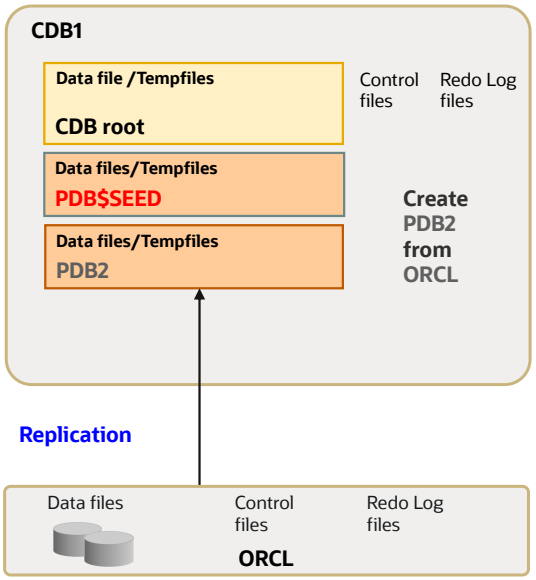
Note: The STATUS of the PDB is CONVERTING.

The technique with the DBMS_PDB package creates an unplugged PDB from an Oracle Database non-CDB. The unplugged PDB can then be plugged into a CDB (of the same version) as a new PDB. To use this technique, the non-CDB must be at release Oracle Database 12*c* or later.

Running the DBMS_PDB.DESCRIBE procedure on the non-CDB generates an XML file that describes the future PDB. You can plug in the unplugged PDB in the same way that you can plug in any unplugged PDB, using the XML file and the non-CDB data files. The steps are the following:

1. Connect to non-CDB ORCL and ensure that the non-CDB ORCL is in read-only mode.
2. Execute the DBMS_PDB.DESCRIBE procedure, providing the file name that will be generated. The XML file contains the list of data files to be plugged. The XML file and the data files described in the XML file comprise an unplugged PDB.
3. Connect to the target CDB to plug the unplugged ORCL as PDB2.
4. Before plugging the unplugged non-CDB, make sure it can be plugged into a CDB by using the DBMS_PDB.CHECK_PLUG_COMPATIBILITY procedure. Execute the CREATE PLUGGABLE command using the clause USING 'XMLfile.xml'. The list of data files from ORCL is read from the XMLfile to locate and name the data files of PDB2.
5. Run the ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql script to delete unnecessary metadata from the PDB SYSTEM tablespace. This script is required for plugging non-CDBs only and must be run before the PDB is opened for the first time.
6. Open PDB2 to verify that the application tables are in PDB2.

Replicating a Non-CDB into a CDB by Using GoldenGate



1. Connect to the CDB root as a common user with the `CREATE PLUGGABLE DATABASE` privilege.
2. Create new **`PDB2`** (from `PDB$SEED`).
3. Open **`PDB2`** in read/write mode.
4. Configure an Oracle GoldenGate unidirectional replication environment from **`ORCL`** to **`PDB2`**.
5. Check application data.

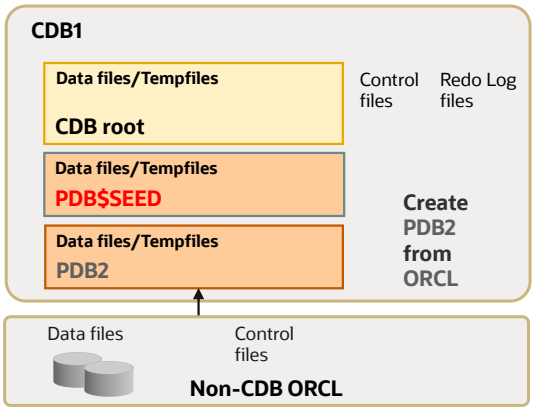
```
SQL> CONNECT sys@PDB2
SQL> SELECT * FROM dba_tables;
SQL> SELECT * FROM HR.EMP;
```

These are the steps for replicating the data from a non-CDB to a PDB by using Oracle GoldenGate replication:

1. Connect to the CDB root as a common user with the `CREATE PLUGGABLE DATABASE` privilege.
2. Create the new `PDB2` from the PDB seed that will be the container for `ORCL` data.
3. Open `PDB2` in read/write mode.
4. Configure an Oracle GoldenGate unidirectional replication environment with the non-CDB `ORCL` as the source database and the `PDB2` as the destination database.
5. When the data at `PDB2` catches up with the data at the non-CDB `ORCL`, switch to `PDB2`.

See *Oracle Database Concepts* for additional information about Oracle GoldenGate.

Cloning a Non-CDB or Remote PDB



PDB_ORCL owns:

- SYSTEM, SYSAUX, UNDO tablespaces
- Full catalog
- A temporary tablespace
- SYS, SYSTEM common users
- New service name

1. Set **ORCL** in READ ONLY mode.
2. Connect to the CDB to create the database link:

```
SQL> CREATE DATABASE LINK link_orcl
             CONNECT TO system IDENTIFIED BY ***
             USING 'orcl';
```

3. Clone the non-CDB:

```
SQL> CREATE PLUGGABLE DATABASE pdb_orcl
             FROM NON$CDB@link_orcl
             CREATE_FILE_DEST = '…/PDB_orcl';
```

4. Run the noncdb_to_pdb.sql script.

```
SQL> CONNECT sys@pdb_orcl AS SYSDBA
SQL> @$ORACLE_HOME/rdbms/admin/noncdb_to_pdb
```

5. Open **PDB_ORCL** in read/write mode.

```
SQL> ALTER PLUGGABLE DATABASE pdb_orcl OPEN;
```

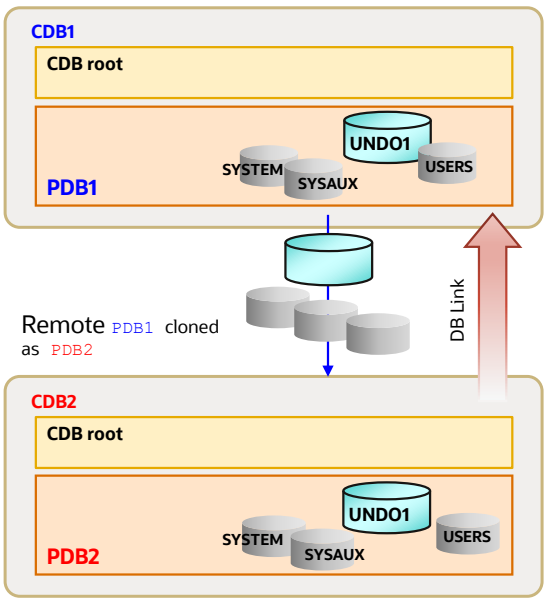
This technique copies a non-CDB or remote PDB and plugs the copy into a CDB.

The steps to clone a non-CDB or remote PDB into a CDB are the following:

1. Set the non-CDB or remote PDB in READ ONLY mode.
2. Connect to the root of the target CDB as a common user with the CREATE PLUGGABLE DATABASE privilege.
3. Create a database link that allows a connection to the remote non-CDB or PDB as a user with the CREATE PLUGGABLE DATABASE privilege.
4. Use the CREATE PLUGGABLE DATABASE command to clone the non-CDB as described in the slide. If you clone a remote PDB, use the source PDB name in place of NON$CDB. Ensure that the new PDB does not conflict with the name of any container within the CDB.
5. If the cloned source is a non-CDB, it is necessary to run the $ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql script.
6. Then open the new PDB with the ALTER PLUGGABLE DATABASE command.
7. Finally, you can re-open the non-CDB or remote PDB.

There are additional clone options such as SNAPSHOT COPY. Refer to the *Oracle Database Administrator's Guide*.

Using DBCA to Clone a Remote PDB



1. Create a common user with privileges in the remote CDB **CDB1**.
2. Use DBCA to clone the remote **PDB1** from **CDB1** to **PDB2**.

```
$ dbca -silent -createPluggableDatabase
-createFromRemotePDB -remotePDBName PDB1
-remoteDBConnString CDB1
-sysDBAUserName system
-sysDBAPassword password
-remoteDBSYSDBAUserName SYS
-remoteDBSYSDBAUserPassword password
-dbLinkUsername c##remote_user
-dbLinkUserPassword password
-sourceDB CDB2 -pdbName PDB2
```

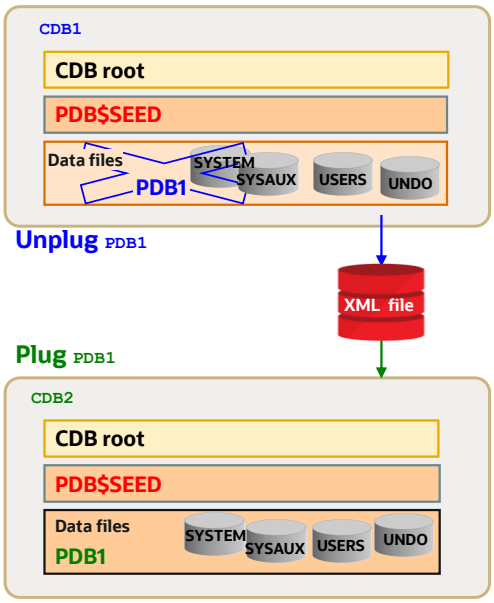
You can use DBCA to clone a remote PDB. The DBCA operation executes the following steps:

1. Checks the presence of the database link. If the database link exists, DBCA drops it.
2. Creates the database link
3. Creates the PDB from the remote PDB
4. Checks the status of the cloned PDB to verify that it is in mounted mode
5. Opens the cloned PDB

The user in the local target CDB must have the `CREATE PLUGGABLE DATABASE` privilege in the CDB root.

The remote CDB must use local undo mode. The remote CDB must be in archivelog mode. The common user in the remote PDB that the database link connects to must have the `CREATE PLUGGABLE DATABASE` and `CREATE SESSION` privileges.

Plugging an Unplugged Regular PDB into CDB



Unplug **PDB1** from **CDB1**:

1. Connect to **CDB1** as a common user.
2. Verify that **PDB1** is closed.

```
SQL> ALTER PLUGGABLE DATABASE pdb1
           UNPLUG INTO 'xmlfile1';
```

3. Drop **PDB1** from **CDB1**.

Plug **PDB1** into **CDB2**:

1. Connect to **CDB2** as a common user.
2. Use the `DBMS_PDB` package to check the compatibility of **PDB1** with **CDB2**.

```
SQL> CREATE PLUGGABLE DATABASE pdb1
             USING 'xmlfile1' NOCOPY;
```

3. Open **PDB1** in read/write mode.

You can create a PDB in a CDB by using the unplugging/plugging method.

Unplugging a PDB disassociates the PDB from a CDB. You unplug a PDB when you want to move the PDB to a different CDB or when you no longer want the PDB to be available.

The first step is to unplug `PDB1` from `CDB1`. The second step is to plug `PDB1` into `CDB2`.

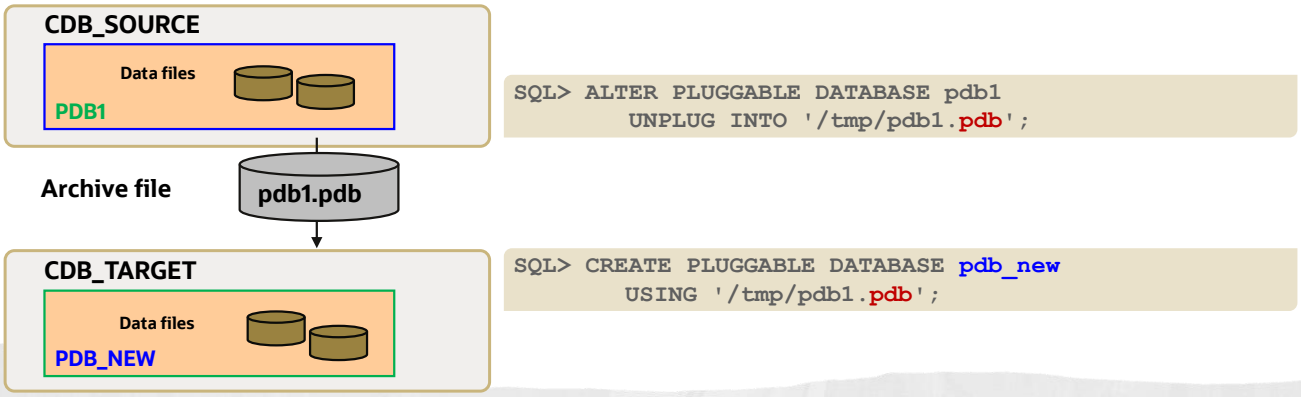
To unplug `PDB1` from `CDB1`, first connect to the source CDB root and check that the PDB is closed using the `V$PDBS` view. Then use `ALTER PLUGGABLE DATABASE` with the `UNPLUG` clause to specify the database to unplug and the XML file to unplug it into. The `STATUS` column in `CDB_PDBS` of the unplugged PDB will be `UNPLUGGED`. A PDB must be dropped from the CDB before it can be plugged back into the same CDB. If the PDB is plugged into another CDB, the PDB does not need to be dropped if the data files are copied.

Before plugging `PDB1` into `CDB2`, you can optionally check whether the unplugged PDB is compatible with `CDB2` by using the `DBMS_PDB.CHECK_PLUG_COMPATIBILITY` function.

To plug `PDB1` into `CDB2`, connect to `CDB2` root and execute the `CREATE PLUGGABLE DATABASE pdb1 USING 'xmlfile1.xml'` command. The last step is to open the PDB.

Plugging in a PDB Using an Archive File

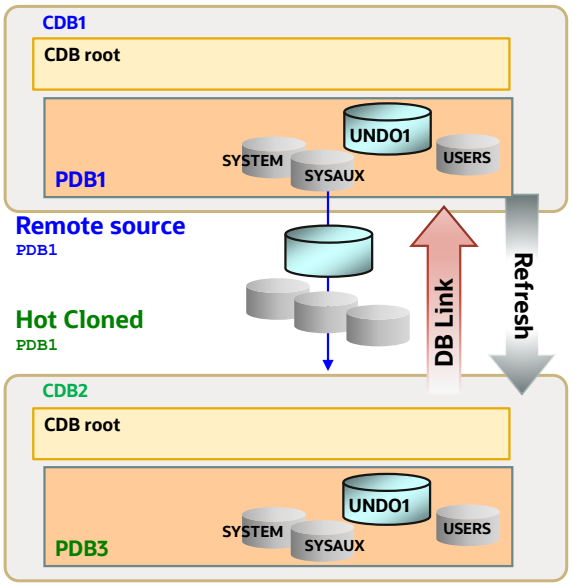
1. Unplugging a PDB into a single archive file includes:
   - XML file
   - Data files
2. Plugging in the PDB requires only the archive file.



When a PDB is unplugged, all the data files associated with the PDB along with the PDB manifest must be copied or moved individually over to the remote server where it will be plugged into another CDB. You can choose to create a single PDB archive file, a compressed file with the `.pdb` extension, which contains the PDB manifest and all the data files. When plugging in a PDB, the presence of a `.pdb` file is interpreted, and the PDB is plugged into the CDB. You can choose to run the PDB plug-in compatibility test directly on the PDB archive without extracting the PDB manifest file from the archive.

This feature provides ease of managing the unplugging and plugging of PDBs across CDBs.

Cloning Remote PDBs in Hot Mode



Remote source PDB still up and fully functional:

1. Connect to the target **CDB2** root to create the database link to **CDB1**.
2. Switch the shared undo mode to local undo mode in both the CDBs.
3. Clone the remote **PDB1** to **PDB3**.
4. Open **PDB3** in read-only or read/write mode.

Incremental refreshing:

- Manual
- Automatic (predefined interval)

Cloning a production PDB to get a test PDB copies the remote production PDB into a CDB while the remote production PDB is still up and fully functional.

Hot remote cloning requires both CDBs to switch from shared undo mode to local undo mode, which means that each PDB uses its own local undo tablespace.

Refreshable Copy

In addition, hot cloning allows incremental refreshing in that the cloned copy of the production database can be refreshed at regular intervals. Incremental refreshing means refreshing an existing clone from a source PDB at a point in time that is more recent than the original clone creation to provide fresh data. A refreshable copy PDB can be opened only in read-only mode.

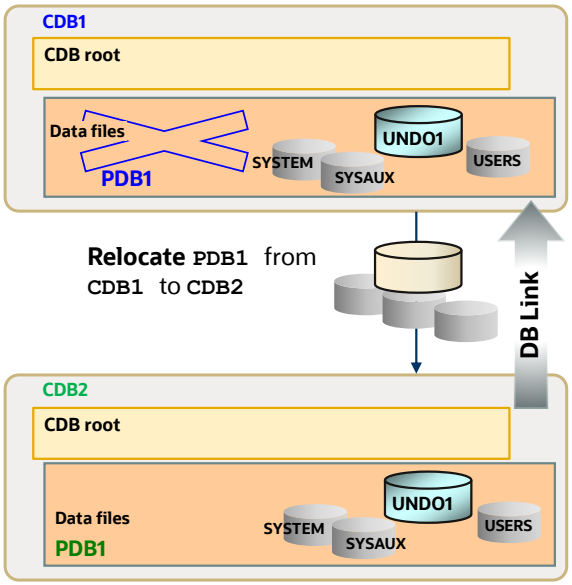
Propagating changes from the source PDB can be performed in two ways:

- Manually (on demand)
- Automatically at predefined time intervals

If the source PDB is not accessible at the moment the refresh copy needs to be updated, archive logs are read from the directory specified by the `REMOTE_RECOVERY_FILE_DEST` parameter to refresh the cloned PDB.

You can also clone an existing PDB by using Database Configuration Assistant (DBCA).

Near-Zero Downtime PDB Relocation



Use a single statement to relocate **PDB1** from **CDB1** into **CDB2**:

1. Switch the shared undo mode to local undo mode in both CDBs.
2. Set ARCHIVELOG mode in both CDBs.
3. Grant `SYSOPER` to the user connected to **CDB1** via the database link created in **CDB2**.
4. Connect to **CDB2** as a common user to create the database link.
5. Use the `CREATE PLUGGABLE DATABASE` statement with the `RELOCATE` clause.
6. Open **PDB1** in read/write mode.

There is no need to:

- Unplug the PDB from the source CDB
- Copy or transfer the data files to a new location
- Plug the PDB in the target CDB
- Drop the source PDB from the source CDB

To get the same result as unplugging and plugging a PDB from a remote source CDB into another CDB, you can take advantage of near-zero downtime PDB relocation.

A single DDL statement can relocate a PDB, using the "pull" mode, connected to the CDB where the PDB will be relocated to pull it from the CDB where the PDB exists, managing existing connections and migrating new connections without requiring any changes to the application.

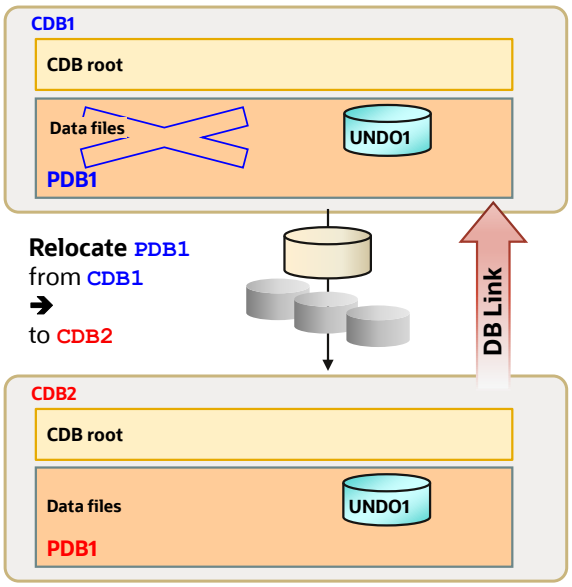
There are two relocation methods:

- **Normal availability mode**
  - When the newly created PDB is opened in read/write mode for the first time, the source PDB is automatically closed and dropped, and the relocation operation is completed with the relocated PDB being fully available. This is the "normal availability" default mode.
  - This method can be used to relocate application PDBs.
- **Maximum availability mode**
  - The maximum availability mode reduces application impact by handling the migration of connections, preserving the source CDB in mount state to guarantee connection forwarding of the listener to the remote listener where the PDB is relocated. In this case, you cannot create a PDB with the same name as the source PDB because it will conflict with the listener forwarding. It is expected that connect strings are updated at a time that is convenient for the application. After this is done and all the clients connect to the new host without forwarding, the DBA can drop the source PDB.

- If `AVAILABILITY MAX` is specified during the `CREATE PLUGGABLE DATABASE RELOCATE` command, additional handling is performed to ensure smooth migration of workload and persistent connection forwarding from the source to the target. The PDB is always first opened in read-only mode. This makes the PDB available as a target for new connections before the source PDB is closed. During this operation, listener information of the target CDB is automatically sent to the source and a special forwarding registration is performed with the source PDB's current listener. New connections to the existing listener are automatically forwarded to connect to the new target. This forwarding persists even after the relocation operation has been completed and effectively allows for no changes to connect strings.
- It is still recommended that connect strings are updated eventually at a time that is convenient for the application, but availability is not dependent on when this action is performed.

PDB relocation requires enabling the local undo mode and ARCHIVELOG mode in both CDBs.

Using DBCA to Relocate a Remote PDB



Use DBCA to relocate the remote PDB1 from CDB1 into CDB2.

```
$ export ORACLE_SID=CDB2
```

```
$ dbca -silent -relocatePDB
-remotePDBName PDB1 -remoteDBConnString CDB1
-sysDBAUserName system
-sysDBAPassword password
-remoteDBSYSDBAUserName SYS
-remoteDBSYSDBAUserPassword password
-dbLinkUsername c##remote_user
-dbLinkUserPassword password
-sourceDB CDB2 -pdbName PDB1
```

You can use DBCA to relocate a remote PDB. The DBCA operation executes the following steps:

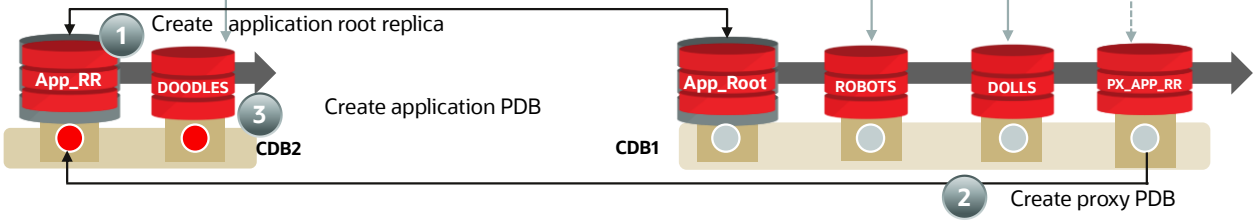
1. Checks the presence of the database link. If the database link exists, DBCA drops it.
2. Creates the database link
3. Creates the PDB from the remote PDB
4. Checks the status of the cloned PDB to verify that it is in mounted mode
5. Opens the relocated PDB

The user in the local database must have the `CREATE PLUGGABLE DATABASE` privilege in the CDB root container. The remote and local databases must be in archivelog mode. The common user in the remote database that the database link connects to must have the `CREATE PLUGGABLE DATABASE`, `SESSION`, and `SYSOPER` privileges.

The local and remote databases must either have the same options installed or the remote database must have a subset of those present on the local database.

Proxy PDB: Query Across CDBs Proxying Root Replica

```
SELECT sum(revenue), year, CDB$NAME, CON$NAME
FROM   CONTAINERS(sales_data)
WHERE  year =  2014  GROUP  BY year, CDB$NAME, CON$NAME;
```



➔ Retrieves rows from the shared table whose data is stored in application PDBs in the application root and replicas in CDBs

Revenue	Year	CDB$NAME	CON$NAME
15000000	2014	CDB1	ROBOTS
20000000	2014	CDB2	DOODLES
10000000	2014	CDB1	DOLLS

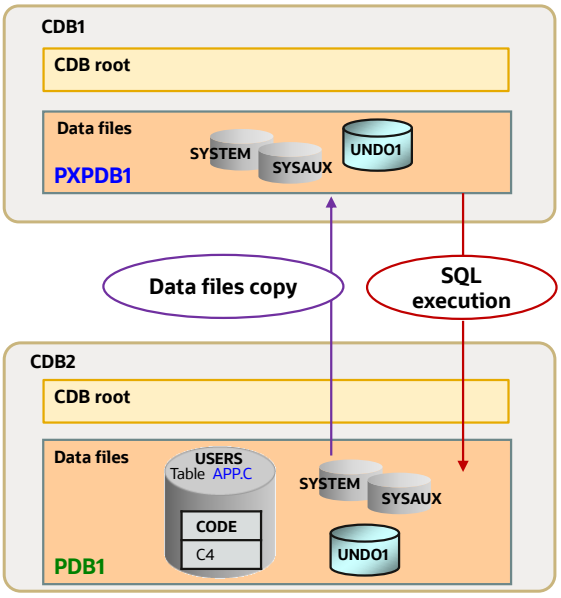
A proxy PDB allows you to execute SQL statements in a remote PDB as if it were a local PDB in the CDB. This type of PDB is very helpful to query data that is spread over PDBs in different CDBs.

In the example in the slide, when connected to the `toys_root` application root, a query on a table shared in the application PDBs, `robots`, `dolls`, and `doodles` cannot retrieve rows from the remote `doodles` if two conditions are not satisfied:

- An application root replica is created in the remote CDB to replicate the application root and, therefore, the application common entities such as tables, users, and privileges.
- A proxy PDB is created in the application root in the local CDB to reference the application root replica in the remote CDB.

When you are connected to the `toys_root` application root and you query from a shared table of the application installed on the `toys_root` application root, the query fetches rows from the two local application PDBs, `robots` and `dolls`, and from the application proxy PDB, `px_app_rr`, executing the query in the remote root replica and, therefore, from the `doodles` application PDB.

Creating a Proxy PDB



A proxy PDB allows execution in a proxied PDB.

1. Switch the shared undo mode to local undo mode in both CDBs.
2. Set the ARCHIVELOG mode in both CDBs.
3. Connect to **CDB1** and create a database link (to **CDB2**).
4. Create the **PXPDB1** proxy PDB in **CDB1** as a view referencing the entire proxied **PDB1** in **CDB2**.

```
SQL> CONNECT sys@cdb1 AS SYSDBA
SQL> CREATE PLUGGABLE DATABASE pxpdb1 AS PROXY
                       FROM pdb1@link_cdb2;
```

5. Execute all the statements in the **PXPDB1** proxy PDB context to have them executed in the proxied **PDB1** PDB in **CDB2**.

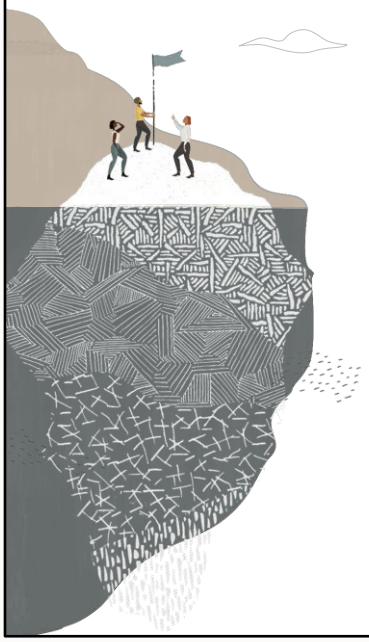
```
SQL> CONNECT sys@pxpdb1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE pxpdb1 OPEN;
SQL> SELECT * FROM app.c;
```

Creating a proxy PDB copies the data files of the `SYSTEM`, `SYSAUX`, and `UNDO` tablespaces of the proxied PDB. A proxy PDB can be created, altered, and dropped from the CDB root like any regular PDB.

The database link must be created in the CDB root that will contain the proxy PDB, and the database link connects either to the remote CDB root or to the remote application container or to the remote application PDB.

Any `ALTER PLUGGABLE DATABASE` statement issued from within the proxy PDB when the PDB is opened is executed in the proxied PDB. The `AS PROXY` clause is used to create a PDB as a proxy PDB. The `IS_PROXY_PDB` column in `CDB_PDBS` displays if a PDB is a proxy PDB.

Summary



- Clone a regular PDB
- Unplug and plug or clone a non-CDB
- Unplug and plug a regular PDB
- Perform hot cloning
- Perform near-zero downtime PDB relocation
- Create and use a proxy PDB

ORACLE
University

Managing PDBs

Objectives

- Change the modes and settings of PDBs
- Evaluate the impact of parameter value changes
- Configure host name and port number per PDB
- Drop PDBs

Changing the PDB Mode

- After closing a PDB, open it in:
  - Restricted read/write mode

```
SQL> CONNECT sys@pdb1 AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE CLOSE;
```

```
SQL> ALTER PLUGGABLE DATABASE OPEN RESTRICTED;
```

```
SQL> SELECT name, open_mode FROM v$pdbs;

NAME                 OPEN_MODE  RES
-------------------- ---------- ---
PDB1                 READ WRITE YES
```

  - Read-only mode

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER PLUGGABLE DATABASE ALL OPEN READ ONLY;
```

You can change the mode of each PDB to perform specific administration operations.

The first example opens the PDB in the `RESTRICTED READ WRITE` mode. This allows only users with the `RESTRICTED SESSION` privilege to connect. This allows the local administrator of the PDB to manage files movement, backups preventing sessions from accessing the data.

Use the `V$PDBS` view to verify that the PDB is in `RESTRICTED READ WRITE` open mode.

The second example opens the PDB in a `READ ONLY` mode. Any session connected to the PDB can perform read-only transactions only.

To change the open mode, first close the PDB. You can apply the same open mode to all PDBs or to only some of them.

Modifying PDB Settings

- Bring a PDB data file online:

```
SQL> ALTER PLUGGABLE DATABASE DATAFILE '/u03/pdb1_01.dbf' ONLINE;
```

- Change the PDB default tablespace:

```
SQL> ALTER PLUGGABLE DATABASE DEFAULT TABLESPACE pdb1_tbs;
```

- Change the PDB default temporary tablespace:

```
SQL> ALTER PLUGGABLE DATABASE DEFAULT TEMPORARY TABLESPACE temp_tbs;
```

- Set the PDB storage limit:

```
SQL> ALTER PLUGGABLE DATABASE STORAGE (MAXSIZE 2G);
```

- Change the global name:

```
SQL> ALTER PLUGGABLE DATABASE RENAME GLOBAL_NAME TO pdbAPP1;
```

You can modify the settings of each PDB without necessarily changing the mode of the PDB. You have to be connected in the PDB to perform the changes in the settings.

The first example uses a `DATAFILE` clause to bring the specified data file online.

The second example sets the default permanent tablespace to `PDB1_TBS` for the PDB.

The third example sets the default temporary tablespace to `TEMP_TBS` for the PDB.

The fourth example sets the storage limit for all tablespaces that belong to the PDB to two gigabytes.

The fifth example changes the global database name of the PDB to `PDBAPP1`. The new global database name for this PDB must be different from that of any container in the CDB. This operation can be done only in restricted mode.

Impact of Changing Initialization Parameters

- A single server parameter file (SPFILE) per CDB
- PDB value changes:
  - Only when `ISPDB_MODIFIABLE=TRUE`
  - Loaded in memory after PDB close
  - Stored in dictionary after CDB shutdown

```
SQL> CONNECT sys@pdb1 AS SYSDBA
Connected.
SQL> ALTER SYSTEM SET ddl_lock_timeout=10;
System altered.
SQL> SHOW PARAMETER ddl_lock_timeout

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
ddl_lock_timeout                     boolean     10
```

There is a single server parameter file (SPFILE) per CDB. Values of the initialization parameters are associated with the CDB root and apply to the CDB root and serve as default values for all other containers.

You can set different parameter values in PDBs when the value of the `ISPDB_MODIFIABLE` column in `V$PARAMETER` is `TRUE`. These are set in the scope of a PDB; then they are remembered properly across PDB close/open and across bouncing the CDB instance. They also travel with clone and unplug/plug operations. Other initialization parameters can be set for the CDB root only.

Connect to the CDB root to view all containers' specific parameter values.

```
SQL> SELECT db_uniq_name, pdb_uid, name, value$ FROM pdb_spfile$ ;


DB_UNIQ_NA  PDB_UID    NAME                                VALUE$
----------  ---------- ----------------------------------- ------
cdb2        3072231663 ddl_lock_timeout                    10
cdb2        4030283986 ddl_lock_timeout                    20
cdb2        3485283967 ddl_lock_timeout                    30
```

Changing Initialization Parameters: Example

```
SQL> CONNECT sys@pdb2 AS SYSDBA

SQL> ALTER SYSTEM SET ddl_lock_timeout=20 SCOPE=BOTH;

SQL> ALTER PLUGGABLE DATABASE CLOSE;
SQL> ALTER PLUGGABLE DATABASE OPEN;
```

```
SQL> CONNECT / AS SYSDBA
SQL> SELECT value, ispdb_modifiable, con_id FROM v$system_parameter
     WHERE  name = 'ddl_lock_timeout';

VALUE                ISPDB     CON_ID
-------------------- ----- ----------
0                    TRUE           0
10                   TRUE           3
20                   TRUE           4
```

In this example, a different value of the DDL_LOCK_TIMEOUT parameter is set in PDB2. The change is maintained after the PDB is closed and reopened. The CON_ID column in the V$SYSTEM_PARAMETER view shows the DDL_LOCK_TIMEOUT value in each container, the CDB root, PDB1, and PDB2.

Using the ALTER SYSTEM Command in a PDB

- Some statements change the way a PDB operates:

ALTER SYSTEM Affecting the PDB only	Objects Impacted
ALTER SYSTEM FLUSH SHARED_POOL	Only for objects of the PDB
ALTER SYSTEM FLUSH BUFFER_CACHE	Only for buffers of the PDB
ALTER SYSTEM ENABLE/DISABLE RESTRICTED SESSION	Only for sessions of the PDB
ALTER SYSTEM KILL SESSION	Only for sessions of the PDB
ALTER SYSTEM SET *parameter*	Only for parameter of the PDB

- Some ALTER SYSTEM statements can be executed in a PDB but affect the whole CDB:

ALTER SYSTEM CHECKPOINT	Affects all data files except those in read only or offline

- All other ALTER SYSTEM statements affect the entire CDB and must be executed by a common user in the CDB root.

ALTER SYSTEM SWITCH LOGFILE	Operation not allowed from within a pluggable database

You can use an ALTER SYSTEM statement to change the way a PDB operates. When the current container is a PDB, you can run the following ALTER SYSTEM statements:

```
ALTER SYSTEM FLUSH SHARED_POOL / BUFFER_CACHE
ALTER SYSTEM ENABLE / DISABLE RESTRICTED SESSION
ALTER SYSTEM SET USE_STORED_OUTLINES
ALTER SYSTEM SUSPEND / RESUME
ALTER SYSTEM CHECK DATAFILES
ALTER SYSTEM REGISTER
ALTER SYSTEM KILL SESSION
ALTER SYSTEM DISCONNECT SESSION
ALTER SYSTEM SET initialization_parameter
```

Some ALTER SYSTEM statements, such as ALTER SYSTEM CHECKPOINT, can be run from within a PDB but actually affect the whole CDB.

Other ALTER SYSTEM statements, such as ALTER SYSTEM SWITCH LOGFILE, affect the entire CDB and must be run by a common user in the CDB root such as ALTER SYSTEM SWITCH LOG unless you set the NONCDB_COMPATIBLE parameter to TRUE. This parameter influences ALTER DATABASE statements behavior in the same way.

Configuring Host Name and Port Number per PDB

- The host name and port number settings for a PDB are important only if proxy PDBs will reference the PDB.

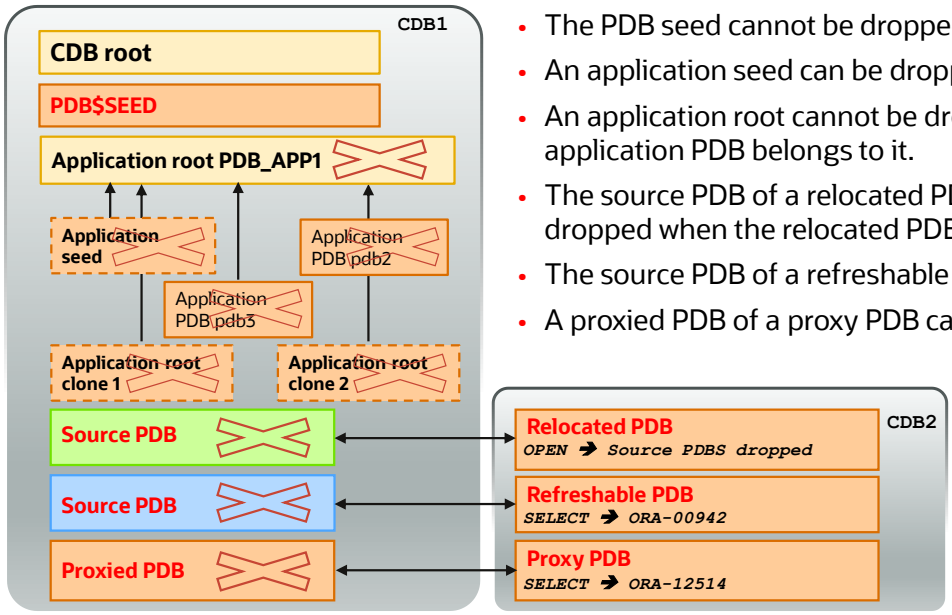
```
SQL> ALTER PLUGGABLE DATABASE CONTAINERS HOST = <host_name>;
SQL> ALTER PLUGGABLE DATABASE CONTAINERS PORT = <port_nb>;
```

- The host name and port number can be reset to their default:

```
SQL> ALTER PLUGGABLE DATABASE CONTAINERS HOST RESET;
SQL> ALTER PLUGGABLE DATABASE CONTAINERS PORT RESET;
```

During PDB creation, the `HOST=<host_name_string>` and `PORT=<port_number>` clauses can be used to indicate the host name and port number to be used by internal database links from proxy PDBs that reference the new PDB. By default, altering the host name and port number for a PDB does not have any effect on internal database links that already have been created to point to this PDB. Only subsequently created internal database links will be affected by the new host name and port number. A proxy PDB will have to be re-created in order for it to pick up the new port number for its target PDB.

Dropping PDBs



- The PDB seed cannot be dropped.
- An application seed can be dropped.
- An application root cannot be dropped as long as an application PDB belongs to it.
- The source PDB of a relocated PDB is automatically dropped when the relocated PDB is opened in RW mode.
- The source PDB of a refreshable PDB can be dropped.
- A proxied PDB of a proxy PDB can be dropped.

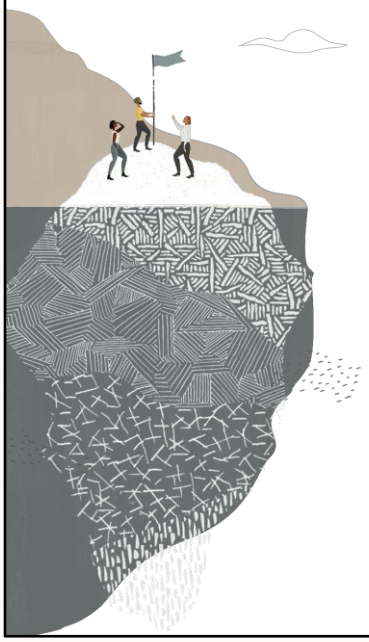
The DROP operation updates controlfiles:

1. Removes PDB data files
2. Retains data files (default)

When you no longer need the data in a PDB, you can drop the PDB. There is only one PDB that cannot be dropped: It is the PDB seed.

- You cannot drop an application root as long as there are still application PDBs associated with it. First drop the application seed and then the application PDBs.
- When the relocation of a PDB is finished, opening the new PDB automatically drops the source PDB.
- Dropping the source PDB of a refreshable PDB does not drop the refreshable PDB. Queries on the source data are no longer possible.
- Dropping the proxied PDB of a proxy PDB does not drop the proxy PDB. Queries on the proxied PDB are no longer possible because the database link to the proxied PDB is no longer valid.

Summary



- Change the different modes and settings of PDBs
- Evaluate the impact of parameter value changes
- Configure host name and port number per PDB
- Drop PDBs

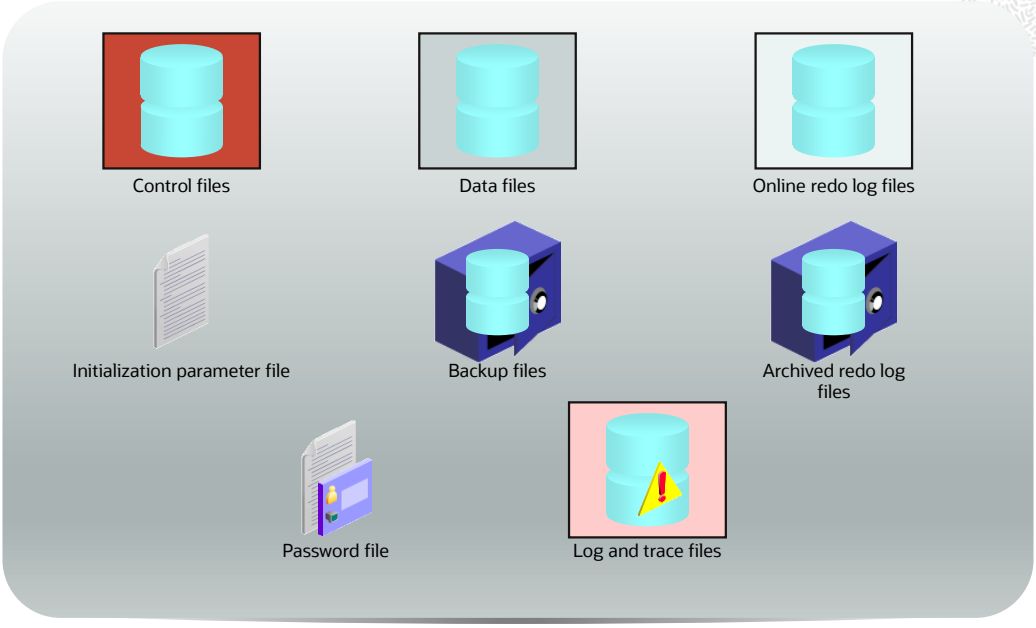
ORACLE
University

Database Storage Overview

Objectives

- Describe logical and physical storage structures in an Oracle database
- Describe the purpose of each of the default tablespaces
- Describe the storage of data in blocks
- List the advantages of deferred segment creation

Database Storage Architecture



The files that comprise an Oracle database are as follows:

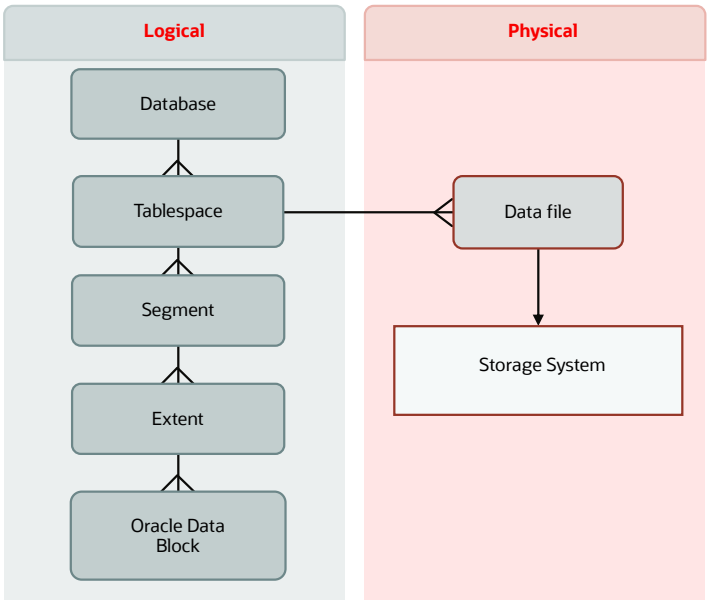
- **Control files:** Each database has one unique control file that contains data about the database itself (that is, physical database structure information). Multiple copies may be maintained to protect against total loss. It can also contain metadata related to backups. The control file is critical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

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

- **Initialization parameter file:** Used to define how the instance is configured when it starts up
- **Password file:** Allows users using the `SYSDBA`, `SYSOPER`, `SYSBACKUP`, `SYSDG`, `SYSKM`, and `SYSASM` roles to connect remotely to the instance and perform administrative tasks
- **Backup files:** Used for database recovery. You typically restore a backup file when a media failure or user error has damaged or deleted the original file.
- **Archived redo log files:** Contain an ongoing history of the data changes (redo) that are generated by the instance. Using these files and a backup of the database, you can recover a lost data file. Archive logs enable the recovery of restored data files.

- **Trace files:** Each server and background process can write to an associated trace file. When an internal error is detected by a process, the process dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, whereas other information is for Oracle Support Services.
- **Alert log file:** These are special trace entries. The alert log of a database is a chronological log of messages and errors. Oracle recommends that you review the alert log periodically.
- **DDL log file:** Contains DDL statements issued by the database server only when the `ENABLE_DDL_LOGGING` initialization parameter is set to `TRUE`

Logical and Physical Database Structures



The database has logical structures and physical structures.

Databases, Tablespaces, and Data Files

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

Tablespaces

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

Data Blocks

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

Extents

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

Segments

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

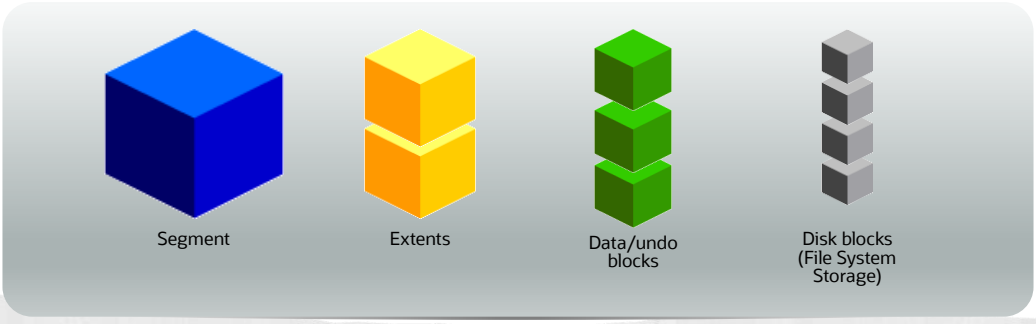
- **Data segments:** Each nonclustered, non-index-organized table has a data segment, except external tables, global temporary tables, and partitioned tables. All the table's data is stored in the extents of its data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment. When deferred segment creation is enabled, segments are not created until the first row is inserted.
- **Index segments:** Each index has an index segment that stores all its data. For a partitioned index, each partition has an index segment.
- **Undo segments:** One `UNDO` tablespace is created for each database instance. This tablespace contains numerous undo segments to temporarily store undo information. The information in an undo segment is used to generate read-consistent database information and, during database recovery, roll back uncommitted transactions for users.
- **Temporary segments:** Temporary segments are created by the Oracle database when a SQL statement needs a temporary work area to complete execution. When the statement finishes execution, the temporary segment's extents are returned to the database for future use. Specify either a default temporary tablespace for every user or a default temporary tablespace that is used databasewide.

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

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

Segments, Extents, and Blocks

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



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

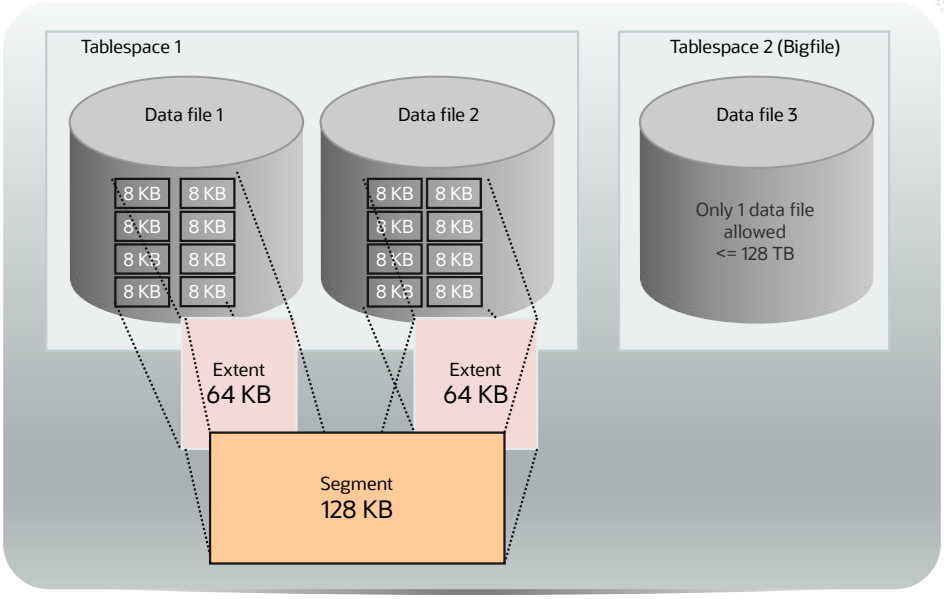
When the database requests a set of data blocks from the operating system (OS), the OS maps this to an actual file system or disk block on the storage device. Because of this, you do not need to know the physical address of any of the data in your database. This also means that a data file can be striped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

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

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

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

Tablespaces and Data Files



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

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

Note: You can also create bigfile tablespaces, which have only one file that is often very large. The file may be of any size up to the maximum that the row ID architecture permits. The maximum size of the single data file or temp file is 128 terabytes (TB) for a tablespace with 32 K blocks and 32 TB for a tablespace with 8 K blocks.

Traditional smallfile tablespaces (which are the default) may contain multiple data files, but the files cannot be as large. For more information about bigfile tablespaces, see *Oracle Database Administrator's Guide*.

Default Tablespaces in a Multitenant Container Database

Tablespace	Description
SYSTEM	The SYSTEM tablespace is used for core functionality. In the root container, it contains Oracle-supplied metadata. In a PDB, it contains user metadata.
SYSAUX	The SYSAUX tablespace is an auxiliary tablespace to the SYSTEM tablespace and helps reduce the load on the SYSTEM tablespace. It exists in the root container and in each PDB.
TEMP	The TEMP tablespace contains schema objects only for a session's duration. There is one default temporary tablespace for the CDB root and one for each application root, application PDB, and PDB.
UNDO	The UNDO tablespace stores the data needed to roll back, or undo, changes to the database. One active undo tablespace exists in the root container. It is recommended that you have a local undo tablespace in each PDB.
USERS	The USERS tablespace stores user objects and data. It is created by default in the root container only.

SYSTEM: It stores the data dictionary (metadata that describes the objects in the database) and tables that contain administrative information about the database. All this information is contained in the SYS schema and can be accessed only by the SYS user or other administrative users with the required privilege. In the root container, it stores Oracle-supplied metadata, whereas in a PDB, it stores user metadata. Pointers from the PDBs to the Oracle-supplied objects allow the “system” objects to be accessed without duplicating them in the PDBs.

SYSAUX: The SYSAUX tablespace is an auxiliary tablespace to the SYSTEM tablespace and helps reduce the load on the SYSTEM tablespace. Some components and products that have used the SYSTEM tablespace or their own tablespaces in earlier releases of Oracle Database now use the SYSAUX tablespace. The SYSAUX tablespace exists in the root container and in each PDB.

TEMP: The TEMP tablespace contains schema objects only for a session's duration. Objects in temporary tablespaces are stored in temp files. Your temporary tablespace is used when you execute a SQL statement that requires the creation of temporary segments (such as a large sort or the creation of an index). Just as each user is assigned a default tablespace for storing created data objects, each user is assigned a temporary tablespace.

UNDO: The UNDO tablespace stores the data needed to roll back, or undo, changes to the database. In a single-instance CDB, one active UNDO tablespace exists in the root container. It is optional, but recommended, to have a local UNDO tablespace in a PDB.

USERS: The USERS tablespace stores user objects and data. If no default tablespace is specified when a user is created, then the USERS tablespace is the default tablespace for all objects created by that user. For the SYS and SYSTEM users, the default permanent tablespace is SYSTEM. The USERS tablespace is created by default in the root container only.

SYSTEM and SYSAUX Tablespaces

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

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

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

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

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

Types of Segments

- A segment is a set of extents allocated for a certain logical structure.
- The different types of segments include:
  - Table and cluster
  - Index
  - Undo
  - Temporary
- Segments are dynamically allocated by the Oracle Database server.

Table and cluster segments: Each nonclustered table has a data segment. All table data is stored in the extents of the table segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.

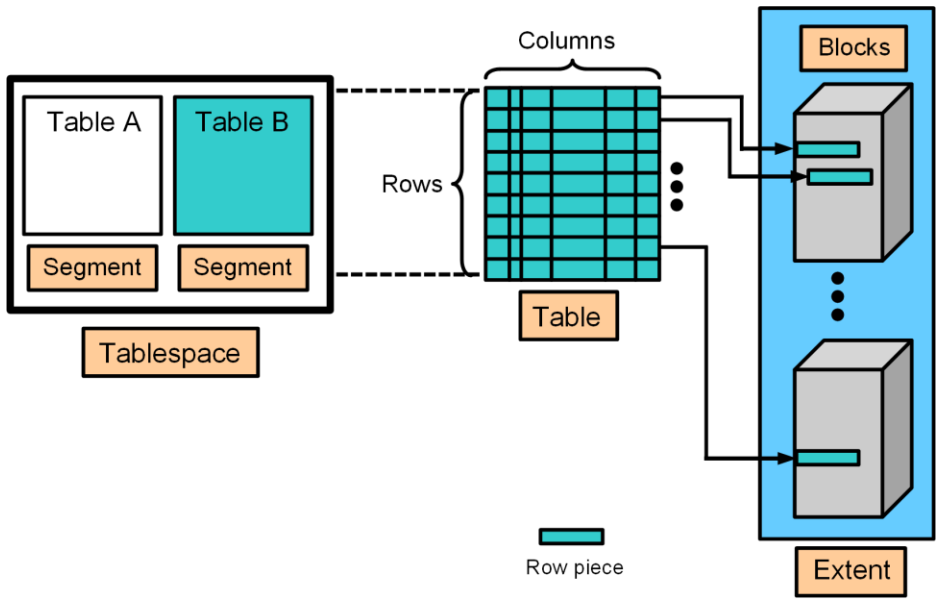
Index segment: Each index has an index segment that stores all its data. For a partitioned index, each partition has an index segment.

Undo segment: Oracle Database maintains information to reverse changes made to the database. This information consists of records of the actions of transactions, collectively known as undo. Undo is stored in undo segments in an undo tablespace.

Temporary segment: A temporary segment is created by the Oracle Database server when a SQL statement needs a temporary database area to complete execution. When the statement finishes execution, the extents in the temporary segment are returned to the system for future use.

The Oracle Database server dynamically allocates space when the existing extents of a segment become full. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

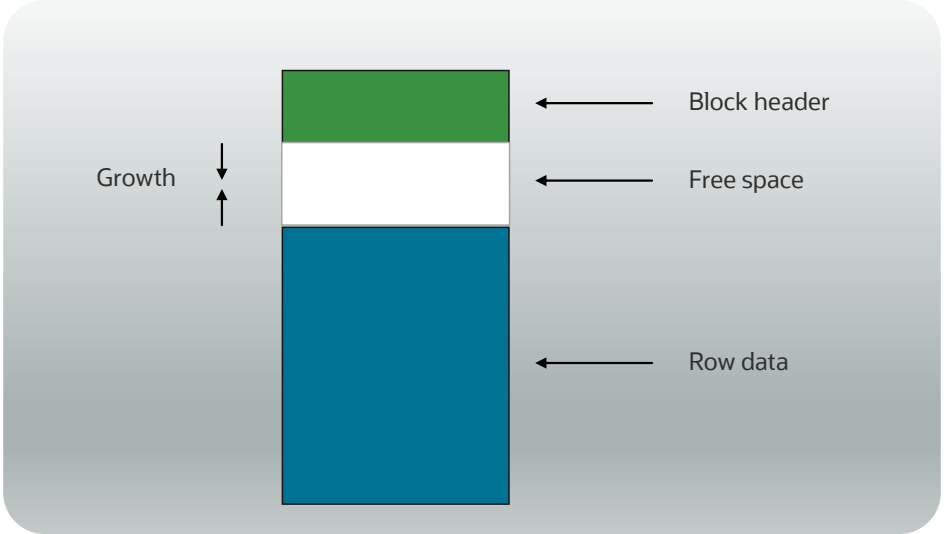
How Table Data Is Stored



Unless deferred segment creation is enabled, a logical "segment" is created when a table is created, as illustrated in the slide. A tablespace contains a collection of segments.

Logically, a table contains rows of column values. A row is ultimately stored in a database block in the form of a row piece (also illustrated in the slide). It is called a row piece because, under some circumstances, the entire row may not be stored in one place. This happens when an inserted row is too large to fit into a single block (chained row) or when an update causes an existing row to outgrow the available free space of the current block (migrated row). Row pieces are also used when a table has more than 255 columns. In this case, the pieces may be in the same block (intra-block chaining) or across multiple blocks.

Database Block Content



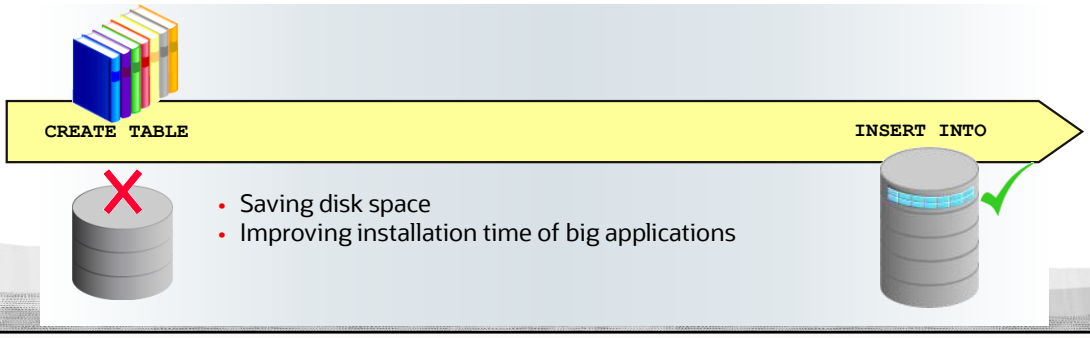
A database block contains a block header, row data, and free space, as illustrated in the slide.

- A block header contains the segment type (such as table or index), data block address, table directory, row directory, and transaction slots of approximately 23 bytes each, which are used when modifications are made to rows in the block. The block header grows downward from the top.
- Row data is the actual data for the rows in the block. Row data space grows upward from the bottom.
- Free space is in the middle of the block, enabling the header and the row data space to grow when necessary. Row data takes up free space as new rows are inserted or as columns of existing rows are updated with larger values. Examples of events that cause header growth:
  - Row directories that need more row entries
  - More transaction slots required than initially configured

Initially, the free space in a block is contiguous. However, deletions and updates may fragment the free space in the block. The free space in the block is coalesced by the Oracle server when necessary.

Understanding Deferred Segment Creation

- DEFERRED_SEGMENT_CREATION = TRUE is the default.
- Deferred segment is the default for tables, indexes, and partitions.
- Segment creation takes place as follows:
  - Table creation > Data dictionary operation
  - DML > Segment creation



When you create a nonpartitioned heap table, table segment creation is deferred to the first row insert. This functionality is enabled by default with the DEFERRED_SEGMENT_CREATION initialization parameter set to TRUE.

Advantages of this space allocation method:

- A significant amount of disk space can be saved for applications that create hundreds or thousands of tables upon installation, many of which might never be populated.
- The application installation time is reduced.

When you insert the first row into the table, the segments are created for the base table, its LOB columns, and its indexes. During segment creation, cursors on the table are invalidated. These operations have a small additional impact on performance.

With this allocation method, it is essential that you do proper capacity planning so that the database has enough disk space to handle segment creation when tables are populated.

Controlling Deferred Segment Creation

- With the DEFERRED_SEGMENT_CREATION parameter:
  - Initialization parameter file
  - ALTER SESSION command
  - ALTER SYSTEM command
- With the SEGMENT CREATION clause:
  - IMMEDIATE
  - DEFERRED (default)

```
CREATE TABLE SEG_TAB3(C1 number, C2 number)
   SEGMENT CREATION IMMEDIATE TABLESPACE SEG_TBS;
CREATE TABLE SEG_TAB4(C1 number, C2 number)
   SEGMENT CREATION DEFERRED;
```

Segment creation can be controlled in two ways:

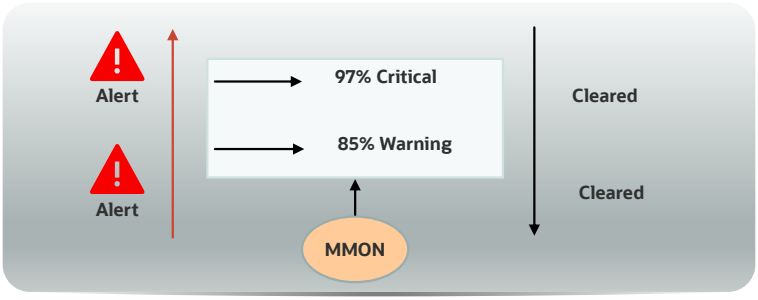
- With the DEFERRED_SEGMENT_CREATION initialization parameter set to TRUE or FALSE. This parameter can be set in the initialization parameter file. You can also control it via the ALTER SESSION or ALTER SYSTEM commands.
- With the SEGMENT CREATION clause of the CREATE TABLE command:
  - SEGMENT CREATION DEFERRED: If specified, segment creation is deferred until the first row is inserted into the table. This is the default behavior.
  - SEGMENT CREATION IMMEDIATE: If specified, segments are materialized during table creation.

This clause takes precedence over the DEFERRED_SEGMENT_CREATION parameter.

It is possible to force the creation of segments for an existing table with the ALTER TABLE … MOVE command.

It is not possible to directly control the deferred segment creation for dependent objects such as indexes. They inherit this characteristic from their parent object—in this case, the table.

Monitoring Tablespace Space Usage



- Read-only and offline tablespaces: Do not set up alerts.
- Temporary tablespace: Threshold corresponds to space currently used by sessions.
- Undo tablespace: Threshold corresponds to space used by active and unexpired extents.
- Auto-extensible files: Threshold is based on the maximum file size.

The database server tracks space utilization while performing regular space management activities. This information is aggregated by the MMON process. An alert is triggered when the threshold for a tablespace has been reached or cleared.

Alerts should not be flagged on tablespaces that are in read-only mode, or tablespaces that were taken offline, because there is not much to do for them.

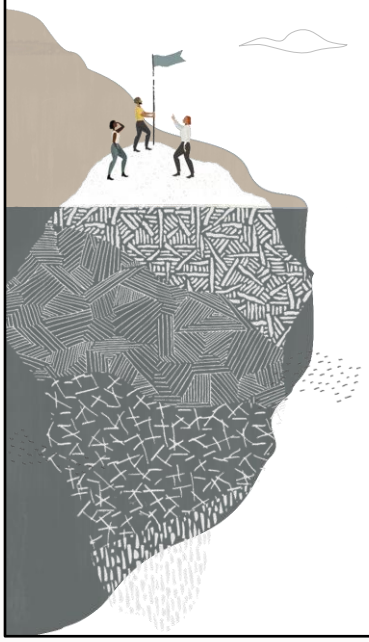
In temporary tablespaces, the threshold value has to be defined as a limit on the used space in the tablespace.

For undo tablespaces, an extent is reusable if it does not contain active or unexpired undo. For the computation of threshold violation, the sum of active and unexpired extents is considered as used space.

For tablespaces with auto-extensible files, the thresholds are computed according to the maximum file size you specified or the maximum OS file size.

The diagram in the slide shows that the MMON process aggregates space utilization information and generates a critical alert when the tablespace is 97% full and a warning when it is 85% full. The alert is cleared after the space usage problem is addressed.

Summary



- Describe logical and physical storage structures in an Oracle database
- Describe the purpose of each of the default tablespaces
- Describe the storage of data in blocks
- List the advantages of deferred segment creation

ORACLE
University

Creating and Managing Tablespaces

Objectives

- Create, alter, and drop tablespaces
- View tablespace information
- Implement Oracle Managed Files (OMF)
- Move and rename online data files

Creating Tablespaces

- A tablespace is an allocation of space in the database that can contain schema objects.
- Create a tablespace with the `CREATE TABLESPACE` statement or use a graphical tool.
- You can create three types of tablespaces:
  - **Permanent tablespace**: Contains persistent schema objects. Objects in permanent tablespaces are stored in data files.
  - **Undo tablespace**: Is a type of permanent tablespace used by Oracle Database to manage undo data in automatic undo management mode
  - **Temporary tablespace**: Contains schema objects only for the duration of a session. Objects in temporary tablespaces are stored in temp files.

Before you can create a tablespace, you must create a database to contain it, and the database must be open. You must also have the `CREATE TABLESPACE` system privilege. To create the `SYSAUX` tablespace, you must have the `SYSDBA` system privilege.

Creating a Tablespace: Clauses

- Include one or more of the following clauses to define various aspects of the tablespace:

Clause	Description
DATAFILE or TEMPFILE	Used to specify the name, location, and initial size of the data file or temp file
ONLINE or OFFLINE	Used to make the tablespace available (or not available) immediately after creation
BLOCKSIZE	Used to specify a nonstandard block size
EXTENT MANAGEMENT	Used to specify how the extents of the tablespace will be managed and where the metadata for allocated and unallocated extents is to be stored
LOGGING	Used to specify the default logging attributes of objects in the tablespace
SEGMENT MANAGEMENT	Used to specify how free space in the segments in the tablespace should be tracked (bitmaps or free lists)

Specifying the File Name and Size

You must include the `DATAFILE` or `TEMPFILE` clause when you create a tablespace. Use this clause to specify the name and location of the data file or temp file. A tablespace must have at least one data file or temp file. You must also specify an initial file size.

You can include the `AUTOEXTEND ON` clause to automatically extend the file when it is full. In this case, you'll need to specify an increment amount and a maximum file size, which can be unlimited. Remember, the size of the file is limited by the physical media on which it resides.

You can include the `BIGFILE` or `SMALLFILE` clause to override the default tablespace type (permanent or temporary) for the database. If you omit this clause, then Oracle Database uses the current default tablespace type.

- A bigfile tablespace contains only one data file (or temp file), which can contain up to approximately 4 billion blocks. Bigfile tablespaces are used with extremely large databases, in which Automatic Storage Management (ASM) or other logical volume managers support the striping or redundant array of independent disks (RAID) and dynamically extensible logical volumes. For bigfile tablespaces, you can specify only one data file in the `DATAFILE` clause or one temp file in the `TEMPFILE` clause.
- A smallfile tablespace is a traditional Oracle tablespace, which can contain 1022 data files or temp files, each of which can contain up to approximately 4 million blocks.

Specifying Tablespace Availability

You can include the `ONLINE` or `OFFLINE` clause to make the tablespace available (or not available) immediately after creation to users who have been granted access to the tablespace. `ONLINE` is the default. The data dictionary view `DBA_TABLESPACES` indicates whether each tablespace is online or offline. This clause cannot be used with temporary tablespaces.

Specifying the Block Size

You can include the BLOCKSIZE clause to specify a nonstandard block size. To specify this clause, the DB_CACHE_SIZE and at least one DB_nK_CACHE_SIZE parameter must be set, and the integer you specify in this clause must correspond with the setting of one DB_nK_CACHE_SIZE parameter setting. You cannot specify nonstandard block sizes for a temporary tablespace or if you intend to assign this tablespace as the temporary tablespace for any users. If you don't specify a block size, the database will use the default 8 KB block size for the tablespace.

Specifying Extent Management

You can include the EXTENT MANAGEMENT clause to specify how the extents of the tablespace will be managed.

- AUTOALLOCATE specifies that the tablespace is system managed. Users cannot specify an extent size. You cannot specify AUTOALLOCATE for a temporary tablespace.
- UNIFORM value specifies that the tablespace is managed with uniform extents of SIZE bytes. The default SIZE is 1MB. All extents of temporary tablespaces are of uniform size, so this keyword is optional for a temporary tablespace. However, you must specify UNIFORM to specify SIZE. You cannot specify UNIFORM for an undo tablespace.

If you don't specify AUTOALLOCATE or UNIFORM, then the default is UNIFORM for temporary tablespaces and AUTOALLOCATE for all other types of tablespaces. If you do not specify the EXTENT_MANAGEMENT clause, then Oracle Database interprets the MINIMUM EXTENT clause and the DEFAULT STORAGE clause to determine extent management.

Also, with the EXTENT MANAGEMENT clause, you can specify where the metadata for allocated and unallocated extents is to be stored, either in the data dictionary (DICTIONARY) or in the tablespace itself (LOCAL). Tablespaces that record extent allocation in the dictionary are called dictionary-managed tablespaces. Tablespaces that record extent allocation in the tablespace header are called locally-managed tablespaces.

Specifying Default Logging Attributes

You can include the LOGGING clause to specify the default logging attributes of all tables, indexes, materialized views, materialized view logs, and partitions within a tablespace. The logging attribute controls whether certain DML operations are logged in the redo log file (LOGGING) or not (NOLOGGING). The default is LOGGING. This clause is not valid for a temporary or undo tablespace.

If logging is not enabled, any direct loads using SQL*Loader and direct load INSERT operations are not written to the redo log, and the objects are thus unrecoverable in the event of data loss. When an object is created without logging enabled, you must back up those objects if you want them to be recoverable. Choosing not to enable logging can have a significant impact on the ability to recover objects in the future. Use with caution.

You can use the FORCE LOGGING clause to put the tablespace into FORCE LOGGING mode. Oracle Database will log all changes to all objects in the tablespace except changes to temporary segments, overriding any NOLOGGING setting for individual objects. The database must be open and in READ WRITE mode.

Specifying Free Space Management

You can include the SEGMENT MANAGEMENT clause to specify whether Oracle Database should track the used and free space in the segments in the tablespace using free lists or bitmaps (AUTO) or not (MANUAL).

- AUTO: The Oracle Database server will use bitmaps to manage the free space in segments. The bitmap describes the status of each data block in a segment with respect to the amount of space in the block that is available for inserting rows. As more or less space becomes available in a data block, the new state is reflected in the bitmap. With bitmaps, the Oracle Database manages free space more automatically. As a result, this form of space management is called Automatic Segment Space Management (ASSM).
- MANUAL: You want to use free lists for managing free space in segments. Free lists are lists of data blocks that have space available for inserting rows. This form of managing space in segments is called manual segment space management because of the need to specify and tune the PCTUSED, FREELISTS, and FREELIST GROUPS storage parameters for schema objects created in the tablespace. This is supported for backward compatibility; it is recommended that you use ASSM.

The SEGMENT MANAGEMENT clause applies to permanent, locally managed tablespaces only and is not valid for temporary tablespaces.

Creating Permanent Tablespaces in a CDB

- Tablespace creation during CDB creation:
  - With DBCA: `USERS` tablespace created in the CDB root
  - With `CREATE DATABASE` statement with `USER_DATA TABLESPACE` clause: Your defined tablespace created in the CDB root
- Create a permanent tablespace in the CDB root:

```
SQL> CONNECT system@cdb1
SQL> CREATE TABLESPACE tbs_CDB_users
           DATAFILE '/u1/app/oracle/oradata/cdb/cdb_users01.dbf' SIZE 100M;
```

- Create a permanent tablespace in a PDB:

```
SQL> CONNECT system@PDB1
SQL> CREATE TABLESPACE tbs_PDB1_users
           DATAFILE '/u1/app/oracle/oradata/cdb/pdb1/users01.dbf' SIZE 100M;
```

In a CDB, one set of tablespaces belong to the CDB root, and each PDB has its own set of tablespaces.

In a multitenant architecture, the tablespace is created in the container where the `CREATE TABLESPACE` command is executed.

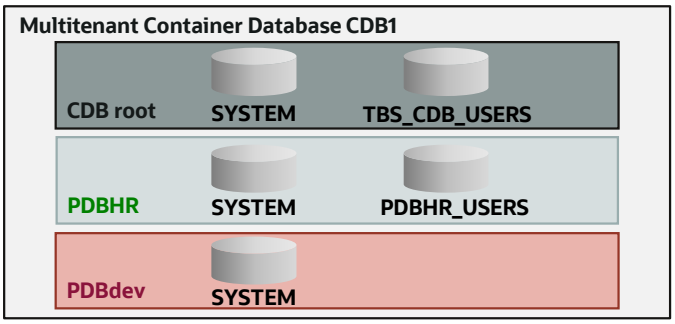
Separating the data files into different directories by PDB can help determine which files belong to which PDB, though it is not necessary.

You can use Oracle ASM storage to manage your disk storage.

The `USER_DATA TABLESPACE` clause in the `CREATE DATABASE` command allows you to specify a default tablespace other than `USERS` when using DBCA to create a database. This tablespace will also be used for XDB options.

Defining Default Permanent Tablespaces

- In the CDB
- In the PDB




```
SQL> CONNECT system@cdb1
SQL> ALTER DATABASE DEFAULT TABLESPACE tbs_CDB_users;
```

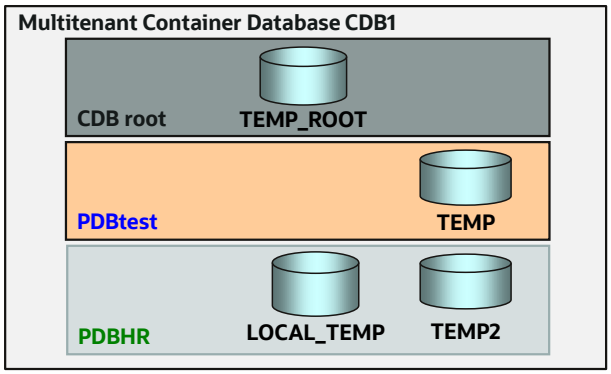
```
SQL> CONNECT pdb1_admin@pdbhr
SQL> ALTER PLUGGABLE DATABASE DEFAULT TABLESPACE pdbhr_users;
```

The default tablespace for a database is a database property. To change the default tablespace for a CDB root container, you must connect to the CDB root container as a user with the proper privileges and issue the `ALTER DATABASE` command. This operation does not change the default permanent tablespace of PDBs.

To change the default tablespace for a PDB, you must connect to the PDB as a user with proper permissions and issue the `ALTER PLUGGABLE DATABASE` command. When connected to the PDB, the `ALTER DATABASE` and `ALTER PLUGGABLE DATABASE` commands perform the same modifications to the PDB. The `ALTER DATABASE` command is allowed for backward compatibility.

Temporary Tablespaces

- Only one default temporary tablespace or tablespace group is allowed per CDB or PDB.
- Each PDB can have temporary tablespaces or tablespace groups.
- Define the default temporary tablespace in a PDB:




```
SQL> CONNECT pdb1_admin@pdbhr
SQL> ALTER DATABASE DEFAULT TEMPORARY TABLESPACE local_temp;
```

A CDB can have only one default temporary tablespace or tablespace group; there can be other temporary tablespaces to which users can be assigned.

PDBs must have their own temporary tablespace (or tablespace group) for use by users in the PDB. These temporary tablespaces will be transported with the PDB when it is unplugged.

Creating a local user requires a default temporary tablespace in the PDB to be assigned.
Creating a common user requires that the default temporary tablespace exists in the container where it is replicated.

The default temporary tablespace for the CDB is set at the CDB root level. There may be multiple temporary tablespaces, but only one can be the default.

When you create a user, you can specify a temporary tablespace to be used by that user. If a temporary tablespace is not specified, the default tablespace for the PDB is used.

The amount of space a PDB can use in the shared temporary tablespace can be limited :

```
ALTER PLUGGABLE DATABASE STORAGE (MAX_SHARED_TEMP_SIZE 500M);
```

In this example, if the value used by sessions that are connected to the PDB is greater than 500M, then no additional storage in the shared temporary tablespace will be available for sessions connected to the PDB until the amount of storage used by them becomes smaller than 500M.

Altering and Dropping Tablespaces

- When you create a tablespace, it is initially a read/write tablespace.
- Use the `ALTER TABLESPACE` statement to take a tablespace offline or online, add data files or temp files to it, or make it a read-only tablespace.
- A tablespace can be in one of three different statuses or states:
  - Read Write
  - Read Only
  - Offline with one of the following options:
    - `NORMAL`
    - `TEMPORARY`
    - `IMMEDIATE`
- Add space to an existing tablespace by either adding data files to the tablespace or changing the size of an existing data file.
- Use the `DROP TABLESPACE` statement to drop a tablespace and its contents from the database if you no longer need its content.

Changing the Status

A tablespace can be in one of three different statuses or states. Any of the following three states may not be available because their availability depends on the type of tablespace:

- **Read Write:** The tablespace is online and can be read from and written to.
- **Read Only:** Specify read-only to place the tablespace in transition read-only mode. In this state, existing transactions can be completed (committed or rolled back), but no further data manipulation language (DML) operations are allowed on the objects in the tablespace. The tablespace is online while in the read-only state. You cannot make the `SYSTEM` or `SYSAUX` tablespaces read-only.
  Note: The undo and temporary tablespaces cannot be made read-only.
- **Offline:** You can take an online tablespace offline so that this portion of the database is temporarily unavailable for general use. The rest of the database is open and available for users to access data. When you take it offline, you can use the following options:
  - **Normal:** A tablespace can be taken offline normally if no error conditions exist for any of the data files of the tablespace. Oracle Database ensures that all data is written to disk by taking a checkpoint for all data files of the tablespace as it takes them offline.
  - **Temporary:** A tablespace can be taken offline temporarily even if there are error conditions for one or more files of the tablespace. Oracle Database takes the data files (which are not already offline) offline, performing checkpointing on them as it does so. If no files are offline, but you use the Temporary clause, media recovery is not required to bring the tablespace back online. However, if one or more files of the tablespace are offline because of write errors, and you take the tablespace offline temporarily, the tablespace requires recovery before you can bring it back online.

- **Immediate:** A tablespace can be taken offline immediately without Oracle Database taking a checkpoint on any of the data files. When you specify Immediate, media recovery for the tablespace is required before the tablespace can be brought online. You cannot take a tablespace offline immediately if the database is running in `NOARCHIVELOG` mode.

Note: System tablespaces may not be taken offline.

Changing the Size

You can add space to an existing tablespace by either adding data files to the tablespace or changing the size of an existing data file. You cannot add additional data files to bigfile tablespaces. You can make a tablespace either larger or smaller. However, you cannot make a data file smaller than the used space in the file. If you try to do so, you'll get an error.

Dropping Tablespaces

You can drop a tablespace and its contents (the segments contained in the tablespace) from the database if the tablespace and its contents are no longer required. You must have the `DROP TABLESPACE` system privilege to drop a tablespace.

When you drop a tablespace, the file pointers in the control file of the associated database are removed. If you are using Oracle Managed Files (OMF), the underlying operating system files are also removed. Otherwise, without OMF, you can optionally direct the Oracle server to delete the operating system files (data files) that constitute the dropped tablespace. If you do not direct the Oracle server to delete the data files at the same time that it deletes the tablespace, you must later use the appropriate commands of your operating system if you want them to be deleted.

You cannot drop a tablespace that contains segments with uncommitted updates from active transactions. For example, if a table in the tablespace is currently being used, or if the tablespace contains undo data that is needed to roll back uncommitted transactions, you cannot drop the tablespace. It is best to take the tablespace offline before dropping it.

Viewing Tablespace Information

- Tablespace and data file information can be obtained by querying the following views:
  - Tablespace information:
    - CDB_TABLESPACES and DBA_TABLESPACES
    - V$TABLESPACE
  - Data file information:
    - CDB_DATA_FILES and DBA_DATA_FILES
    - V$DATAFILE
  - Temp file information:
    - CDB_TEMP_FILES and DBA_TEMP_FILES
    - V$TEMPFILE
  - Tables in a tablespace:
    - ALL_TABLES

V$TABLESPACE, V$DATAFILE, and V$TEMPFILE display information from the control file.

Implementing Oracle Managed Files (OMF)

- Specify file operations in terms of database objects rather than file names.

Parameter	Description
DB_CREATE_FILE_DEST	Defines the location of the default file system directory for data files and temporary files
DB_CREATE_ONLINE_LOG_DEST_n	Defines the location for redo log files and control file creation
DB_RECOVERY_FILE_DEST	Gives the default location for the fast recovery area

- Example:

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/u01/app/oracle/oradata';
SQL> CREATE TABLESPACE tbs_1;
```

Oracle Managed Files (OMF) eliminates the need for you to directly manage the operating system files in an Oracle database. You specify operations in terms of database objects rather than file names. The database internally uses standard file system interfaces to create and delete files as needed for the following database structures:

- Tablespaces
- Redo log files
- Control files
- Archived logs
- Block change tracking files
- Flashback logs
- RMAN backups

A database can have a mixture of Oracle-managed and Oracle-unmanaged files. The file system directory specified by either of these parameters must already exist; the database does not create it. The directory must also have permissions for the database to create the files in it.

The table in the slide lists three initialization parameters that are used with OMF. The example in the slide shows that after the `DB_CREATE_FILE_DEST` initialization parameter is set, you can omit the `DATAFILE` clause from the `CREATE TABLESPACE` statement. The data file is created in the location specified by `DB_CREATE_FILE_DEST`.

Naming Formats

Oracle-managed files have a specific naming format. For example, on Linux- and UNIX-based systems, the following format is used:

```
<destination_prefix>/o1_mf_%t_%u_.dbf
```

Do not rename an Oracle-managed file. The database identifies an Oracle-managed file based on its name. If you rename the file, the database is no longer able to recognize it as an Oracle-managed file and will not manage the file accordingly.

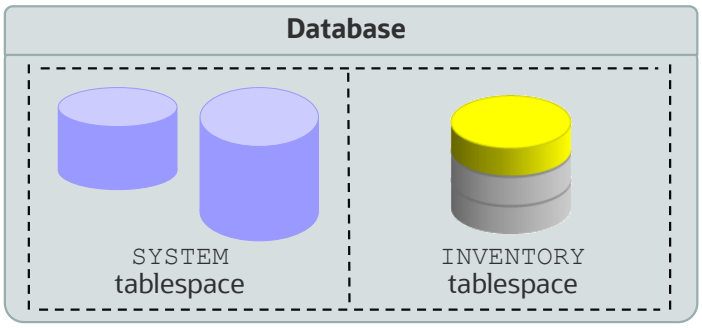
File Size

By default, Oracle-managed data files, including those for the `SYSTEM` and `SYSAUX` tablespaces, are 100 MB and auto-extensible.

Note: By default, Automatic Storage Management (ASM) uses OMF files, but if you specify an alias name for an ASM data file at tablespace creation time or when adding an ASM data file to an existing tablespace, then that file will not be OMF.

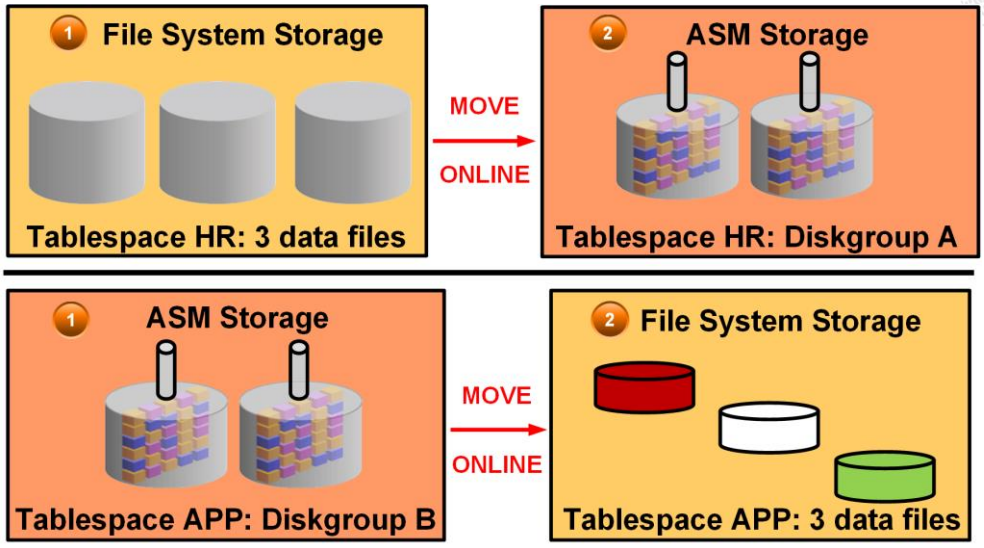
Enlarging the Database

- You can enlarge the database in the following ways:
  - Create a new tablespace.
  - Add a data file to an existing smallfile tablespace.
  - Increase the size of a data file.
  - Provide for the dynamic growth of a data file.



These activities can be performed with Enterprise Manager or with SQL statements. The size of the database can be described as the sum of all of its tablespaces.

Moving or Renaming Online Data Files



You can rename and move an online data file from one kind of storage system to another while the database is open and accessing the file. The first example in the diagram illustrates moving the `HR` tablespace (three data files) from file system storage to ASM storage (diskgroup A). The second example illustrates moving the `APP` tablespace from ASM storage (diskgroup B) to file system storage (three data files).

Queries and DML and DDL operations can be performed while the data file is being moved. For example:

- `SELECT` statements against tables and partitions
- Creation of tables and indexes
- Rebuilding of indexes

Other notes:

- If objects are compressed while the data file is moved, the compression remains the same.
- You do not have to shut down the database or take the data file offline while you move a data file to another location, disk, or storage system.
- You can omit the `TO` clause only when an Oracle-managed file is used. In this case, the `DB_CREATE_FILE_DEST` initialization parameter should be set to indicate the new location.
- If the `REUSE` option is specified, the existing file is overwritten.
- If the `KEEP` clause is specified, the old file will be kept after the move operation. The `KEEP` clause is not allowed if the source file is an Oracle-managed file.
- Use the `V$SESSION_LONGOPS` view to display ongoing online move operations. Each ongoing operation has one row. The state transition of a successful online move operation is usually `NORMAL` to `COPYING` to `SUCCESS` and finally to `NORMAL`.

Examples: Moving and Renaming Online Data Files

- Relocating an online data file:

```
SQL> ALTER DATABASE MOVE DATAFILE '/disk1/myexample01.dbf'
  2  TO '/disk2/myexample01.dbf';
```

- Copying a data file from a file system to Automatic Storage Management (ASM):

```
SQL> ALTER DATABASE MOVE DATAFILE '/disk1/myexample01.dbf'
  2  TO '+DiskGroup2' KEEP;
```

- Renaming an online data file:

```
SQL> ALTER DATABASE MOVE DATAFILE '/disk1/myexample01.dbf`
  2  TO '/disk1/myexample02.dbf';
```

You do not have to shut down the database or take the data file offline while you move a data file to another location, disk, or storage system.

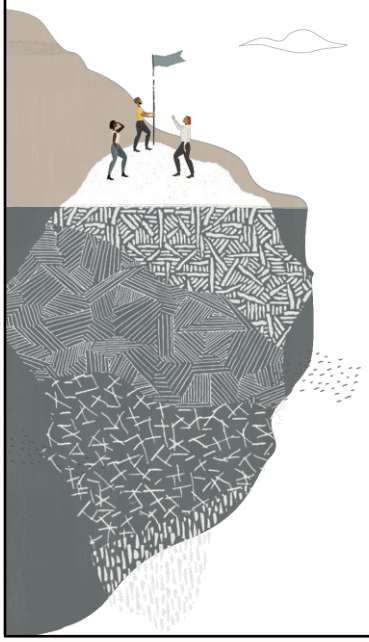
The `TO` clause can be omitted only when an Oracle-managed file is used. In this case, the `DB_CREATE_FILE_DEST` parameter should be set to indicate the new location.

If the `REUSE` option is specified, the existing file is overwritten.

If the `KEEP` clause is specified, the old file will be kept after the move operation. The `KEEP` clause is not allowed if the source file is an Oracle-managed file.

Use the `V$SESSION_LONGOPS` view to display ongoing online move operations. Each ongoing operation has one row. The state transition of a successful online move operation is usually `NORMAL` to `COPYING` to `SUCCESS` and finally to `NORMAL`.

Summary



- Create, alter, and drop tablespaces
- View tablespace information
- Implement Oracle Managed Files (OMF)
- Move and rename online data files

ORACLE
University

Improving Space Usage

Objectives

- Describe and use Oracle Database features that save space
- Create private temporary tables
- Save space by using compression
- Reclaim wasted space from tables and indexes by using the segment shrink functionality
- Manage resumable space allocation

Space Management Features

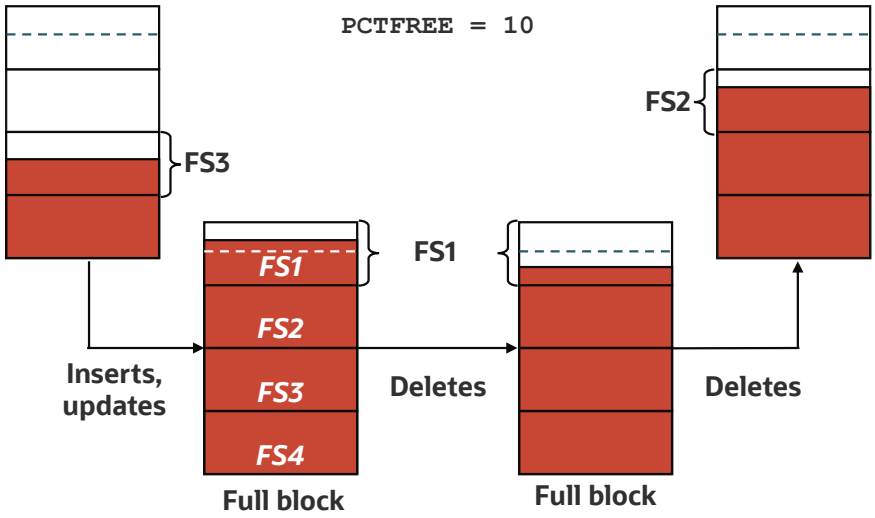
- Space is automatically managed by the Oracle Database server. It generates alerts about potential problems and recommends possible solutions.
- Space management features include:
  - Oracle Managed Files (OMF)
  - Free-space management with bitmaps ("locally managed") and automatic data file extension
  - Proactive space management (default thresholds and server-generated alerts)
  - Space reclamation (shrinking segments, online table redefinition)
  - Capacity planning (growth reports)

With Oracle Managed Files (OMF), you can specify operations in terms of database objects rather than file names. The Oracle Database server can manage free space within a tablespace with bitmaps. This is known as a "locally managed" tablespace. In addition, free space within segments located in locally managed tablespaces can be managed using bitmaps. This is known as Automatic Segment Space Management. The bitmapped implementation eliminates most space-related tuning of tables, while providing improved performance during peak loads. Additionally, the Oracle Database server provides automatic extension of data files, so the files can grow automatically based on the amount of data in the files.

When you create a database, proactive space monitoring is enabled by default (this causes no performance impact). The Oracle Database server monitors space utilization during normal space allocation and deallocation operations and alerts you if the free space availability falls below the predefined thresholds (which you can override). Advisors and wizards assist you with space reclamation.

For capacity planning, the Oracle Database server provides space estimates based on table structure and the number of rows and a growth trend report based on historical space utilization stored in the Automatic Workload Repository (AWR).

Block Space Management



Space management involves the management of free space at the block level. With Automatic Segment Space Management, each block is divided into four sections, named FS1 (between 0 and 25% of free space), FS2 (25% to 50% free), FS3 (50% to 75% free), and FS4 (75% to 100% free).

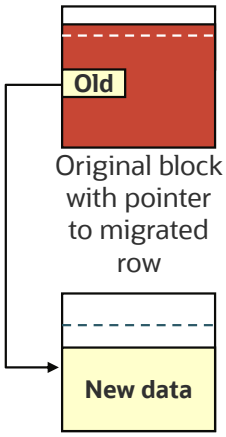
Depending on the level of free space in the block, its status is automatically updated. That way, depending on the length of an inserted row, you can tell whether a particular block can be used to satisfy an insert operation. Note that a "full" status means that a block is no longer available for inserts.

In the example in the slide, the block on the left is an FS3 block because it has between 50% and 75% free space. After some insert and update statements, `PCTFREE` is reached (the dashed line), and it is no longer possible to insert new rows in that block. The block is now considered as a "full" or FS1 block. The block is considered for insertion again as soon as its free space level drops below the next section. In the preceding case, it gets FS2 status as soon as the free space is more than 25%.

Note: Large object (`LOB`) data types (`BLOB`, `CLOB`, `NCLOB`, and `BFILE`) do not use the `PCTFREE` storage parameter. Uncompressed and OLTP-compressed blocks have a default `PCTFREE` value of 10; basic compressed blocks have a default `PCTFREE` value of 0.

Row Chaining and Migration

- On update: Row length increases, exceeding the available free space in the block.
- Data needs to be stored in a new block.
- Original physical identifier of row (ROWID) is preserved.
- The Oracle Database server needs to read two blocks to retrieve data.
- Segment Advisor finds segments containing the migrated rows.
- There is automatic coalescing of fragmented free space inside the block.



In two circumstances, the data for a row in a table may be too large to fit into a single data block. In the first case, the row is too large to fit into one data block when it is first inserted. In this case, the Oracle Database server stores the data for the row in a chain of data blocks (one or more) reserved for that segment. Row chaining most often occurs with large rows, such as rows that contain a column of data type LONG or LONG RAW. Row chaining in these cases is unavoidable.

However, in the second case, a row that originally fit into one data block is updated so that the overall row length increases, and the block's free space is already completely filled. In this case, the Oracle Database server migrates the data for the entire row to a new data block, assuming that the entire row can fit in a new block. The database preserves the original row piece of a migrated row to point to the new block containing the migrated row. The ROWID of a migrated row does not change.

When a row is chained or migrated, input/output (I/O) performance associated with this row decreases because the Oracle Database server must scan more than one data block to retrieve the information for the row.

Segment Advisor finds the segments containing migrated rows that result from an UPDATE.

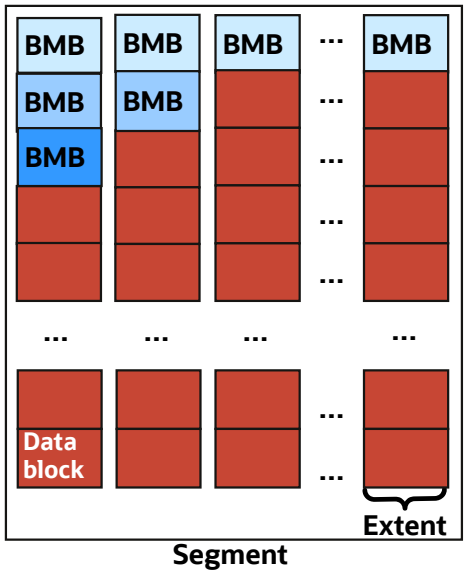
The Oracle Database server automatically and transparently coalesces the free space of a data block when:

- An INSERT or UPDATE statement attempts to use a block with sufficient free space for a new row piece
- The free space is fragmented so that the row piece cannot be inserted in a contiguous section of the block

After coalescing, the amount of free space is identical to the amount before the operation, but the space is now contiguous.

Free Space Management Within Segments

- Tracked by bitmaps in segments
- Benefits:
  - More flexible space utilization
  - Runtime adjustment
  - Multiple process search of bitmap blocks (BMBs)



Free space can be managed automatically inside database segments. The in-segment free or used space is tracked with bitmaps. To take advantage of this feature, specify Automatic Segment Space Management when you create a locally managed tablespace. Your specification then applies to all segments subsequently created in this tablespace.

Automatic space management segments have a set of bitmap blocks (BMBs) describing the space utilization of the data blocks in that segment. BMBs are organized in a tree hierarchy. The root level of the hierarchy, which contains references to all intermediate BMBs, is stored in the segment header. The leaves of this hierarchy represent the space information for a set of contiguous data blocks that belong to the segment. The maximum number of levels inside this hierarchy is three.

Benefits of using automatic space management include:

- Better space utilization, especially for objects with highly varying row sizes
- Better runtime adjustment to variations in concurrent access
- Better multi-instance behavior in terms of performance or space utilization

Allocating Extents

- Searching the data file's bitmap for the required number of adjacent free blocks
- Sizing extents with storage clauses:
  - `UNIFORM`
  - `AUTOALLOCATE`
- Viewing the extent map
- Obtaining deallocation advice

With locally managed tablespaces, the Oracle Database server looks for free space to allocate to a new extent by first determining a candidate data file in the tablespace and then searching the data file's bitmap for the required number of adjacent free blocks. If that data file does not have enough adjacent free space, then the Oracle Database server looks in another data file.

Two clauses affect the sizing of extents:

- With the `UNIFORM` clause, the database creates all extents of a uniform size that you specified (or a default size) for any objects created in the tablespace.
- With the `AUTOALLOCATE` clause, the database determines the extent-sizing policy for the tablespace.

The Oracle Database server provides a Segment Advisor that helps you determine whether an object has space available for reclamation on the basis of the level of space fragmentation within the object.

Using Unusable Indexes

- Consider using unusable indexes to improve the performance of bulk loads.
- Unusable indexes are ignored by the optimizer.
- When an unusable index is created, no segment is created:

```
CREATE INDEX test_i1 ON seg_test(c) UNUSABLE
```

- When an existing index is altered to unusable, the segment is dropped:

```
ALTER INDEX test_i UNUSABLE
```

- An unusable index can be rebuilt to make it valid again:

```
ALTER INDEX test_i REBUILD
```

You can use an unusable index to improve the performance of bulk load.

When a new index is created with the `UNUSABLE` attribute, the index is defined but no segment is created. DML operations do not cause any updates to the index, and the optimizer does not use it.

You can also set an existing index to unusable. When you do this, the index segment is dropped. You can re-create the index by using the `REBUILD` option on the `ALTER INDEX` statement.

Using Temporary Tables

- Temporary tables contain data for the duration of a transaction or session.
- Types of temporary tables:
  - **Global**: Table definition is visible to all sessions; content is specific to a session.
  - **Private**: Table definition is visible only to the creating session.
- Segment for a temporary table is allocated with the first `INSERT` or `CREATE TABLE AS SELECT` statement.
- Table definition persists after a `ROLLBACK`.
- Transaction-specific temporary table can only be used by one transaction at a time.

Consider using temporary tables in applications where a result set is to be buffered (temporarily persisted). You can create either a global temporary table or a private temporary table.

A global temporary table definition is visible to all sessions. However, the content in the table is specific to a session. Global temporary table definitions are stored on disk.

A private temporary table definition is visible only to the session that created it. The table definition is stored only in memory.

Creating Global Temporary Tables

- Create a global temporary table by using the `CREATE GLOBAL TEMPORARY TABLE` statement.
- Specify whether the global temporary table applies to a transaction or session by using the `ON COMMIT` clause:
  - Transaction-specific (default): `ON COMMIT DELETE ROWS`
  - Session-specific: `ON COMMIT PRESERVE ROWS`
- Example:

```
SQL> CREATE GLOBAL TEMPORARY TABLE trans_buff_area(date1 DATE,…)
   > ON COMMIT DELETE ROWS;
```

You can use the `CREATE GLOBAL TEMPORARY TABLE` statement to create a global temporary table. The `ON COMMIT` clause indicates if the data in the table is transaction-specific (the default) or session-specific.

If the global temporary table is transaction-specific, the table is truncated after each commit. With a session-specific global temporary table, the table is truncated when the session is terminated.

Creating Private Temporary Tables

- Create a private temporary table by using the `CREATE PRIVATE TEMPORARY TABLE` statement.
- The table name must start with `ORA$PTT_`:

```
PRIVATE_TEMP_TABLE_PREFIX = ORA$PTT_
```

```
SQL> CREATE PRIVATE TEMPORARY TABLE ORA$PTT_mine (c1 DATE, … c3 NUMBER(10,2));
```

- The `CREATE PRIVATE TEMPORARY TABLE` statement does not commit a transaction.
- Two concurrent sessions may have a private temporary table with the same name but different shape.
- Private temporary table definition and contents are automatically dropped at the end of a session or transaction.

```
SQL> CREATE PRIVATE TEMPORARY TABLE ORA$PTT_mine (c1 DATE …)
             ON COMMIT PRESERVE DEFINITION;
```

```
SQL> DROP TABLE ORA$PTT_mine;
```

Private temporary tables are local to a specific session. In contrast with global temporary tables, the definition and contents are local to the creating session only and are not visible to other sessions.

There are two types of durations for private temporary tables:

- **Transaction:** The private temporary table is automatically dropped when the transaction in which it was created ends with either a `ROLLBACK` or `COMMIT`. This is the default behavior if no `ON COMMIT` clause is defined at private temporary table creation.
- **Session:** The private temporary table is automatically dropped when the session that created it ends. This is the behavior if the `ON COMMIT PRESERVE DEFINITION` clause is defined at the private temporary table creation.

A private temporary table must be named with a prefix '`ORA$PTT_`'. The prefix is defined by default by the `PRIVATE_TEMP_TABLE_PREFIX` initialization parameter, modifiable at the instance level only.

Creating a private temporary table does not commit the current transaction. Because it is local to the current session, a concurrent session may also create a private temporary table with the same name but having a different shape.

Private temporary tables must be created in the user schema. Creating a private temporary table in another schema is not allowed.

Table Compression: Overview

- Reducing storage costs by compressing all data:
  - Basic compression for direct-path insert operations: 10x
  - Advanced row compression for all DML operations: 2–4x

Compression Method	Compression Ratio	CPU Overhead	CREATE and ALTER TABLE Syntax	Typical Applications
Basic table compression	High	Minimal	COMPRESS or ROW STORE COMPRESS BASIC	DSS
Advanced row compression	High	Minimal	ROW STORE COMPRESS ADVANCED	OLTP, DSS

Oracle Database supports three methods of table compression:

- Basic table compression
- Advanced row compression
- Hybrid columnar compression

Oracle Corporation recommends compressing all data to reduce storage costs. The Oracle Database server can use table compression to eliminate duplicate values in a data block. For tables with highly redundant data, compression saves disk space and reduces memory use in the database buffer cache. Table compression is transparent to database applications.

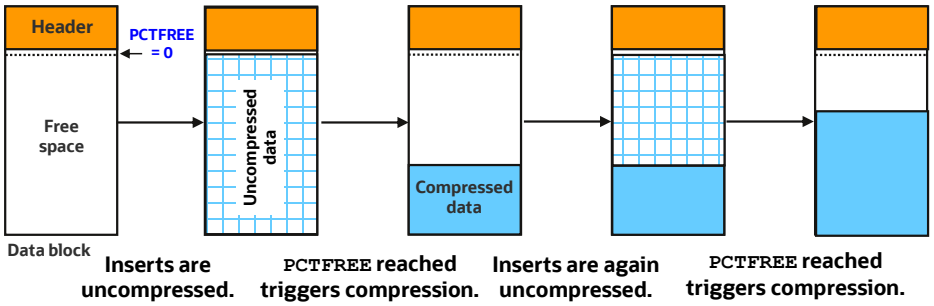
The TABLE_COMPRESSION clause is valid only for heap-organized tables. The COMPRESS keyword enables table compression. The NOCOMPRESS keyword disables table compression. NOCOMPRESS is the default.

With basic compression, the Oracle Database server compresses data at the time of performing bulk load by using operations such as direct loads or CREATE TABLE AS SELECT.

With ROW STORE COMPRESS ADVANCED, the Oracle Database server compresses data during all DML operations on the table.

Hybrid Columnar Compression is optimized for data warehousing and decision support applications on Oracle Exadata storage. Other Oracle storage systems support Hybrid Columnar Compression and deliver the same space savings as on Oracle Exadata storage, but do not deliver the same level of query performance.

Table Compression: Concepts



The slide shows you a data block evolution when that block is part of a compressed table. You should read it from left to right. At the start, the block is empty and available for inserts. When you start inserting into this block, data is stored in an uncompressed format (as for uncompressed tables). However, as soon as the block is filled based on the `PCTFREE` setting of the block, the data is automatically compressed, potentially reducing the space it originally occupied. This allows for new uncompressed inserts to take place in the same block, until it is once again filled based on the `PCTFREE` setting. At that point compression is triggered again to reduce the amount of space used in the block. Compression eliminates holes created due to deletions and maximizes contiguous free space in blocks.

Tables with `COMPRESS` or `ROW STORE COMPRESS BASIC` use a `PCTFREE` value of `0` to maximize compression, unless you explicitly set a value for the `PCTFREE` clause.

Tables with `ROW STORE COMPRESS ADVANCED` or `NOCOMPRESS` use the `PCTFREE` default value of `10` to maximize compression while still allowing for some future DML changes to the data, unless you override this default explicitly.

Compression for Direct-Path Insert Operations

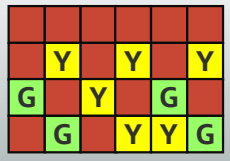
- Is enabled with `CREATE TABLE ... COMPRESS BASIC`
- Is recommended for bulk loading data warehouses
- Maximizes contiguous free space in blocks

Enable basic table compression by using `COMPRESS` or `ROW STORE COMPRESS BASIC`. The Oracle Database server attempts to compress data during the following direct-path insert operations when it is productive to do so:

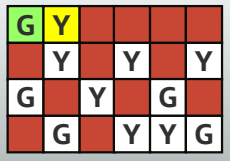
- Direct-path SQL*Loader
- `CREATE TABLE AS SELECT` statements
- Parallel `INSERT` statements
- `INSERT` statements with an `APPEND` hint

Advanced Row Compression for DML Operations

- Is enabled with `CREATE TABLE ... ROW STORE COMPRESS ADVANCED`
- Is recommended for active OLTP environments



Uncompressed block



OLTP compression with the symbol table at the beginning of the block

Enable advanced row compression by using `ROW STORE COMPRESS ADVANCED`

The Oracle Database server compresses data during all DML operations on the table. This form of compression is recommended for active OLTP environments.

With advanced row compression, duplicate values in the rows and columns in a data block are stored once at the beginning of the block in a symbol table. Duplicate values are replaced with a short reference to the symbol table, as shown in the diagram in the slide. Thus, information needed to re-create the uncompressed data is stored in the block.

To illustrate the principle of advanced row compression, the diagram shows two rectangles. The first red rectangle contains four small green squares labeled “G” and six yellow ones labeled “Y.” They represent uncompressed blocks. At the beginning of the second red rectangle, there is only one green square labeled “G” and one yellow “Y” square, representing the symbol table. The second red diagram shows 10 white squares in the same position as the green and yellow ones. They are white because they are now only a reference, not consuming space for duplicate values.

Specifying Table Compression

- You can specify table compression for:
  - An entire heap-organized table
  - A partitioned table (each partition can have a different type or level of compression)
  - The storage of a nested table
- You cannot:
  - Specify basic and advanced row compression on tables with more than 255 columns
  - Drop a column if a table is compressed for direct loads, but you can drop it if the table is advance row compressed

Table compression has the following restrictions:

- `ROW STORE COMPRESS ADVANCED` and `COMPRESS BASIC` are not supported for tables with more than 255 columns.
- You cannot drop a column from a table that is compressed for direct-load operations, although you can set such a column as unused. All the operations of the `ALTER TABLE ... drop_column_clause` are valid for tables that are compressed for OLTP.

Using the Compression Advisor

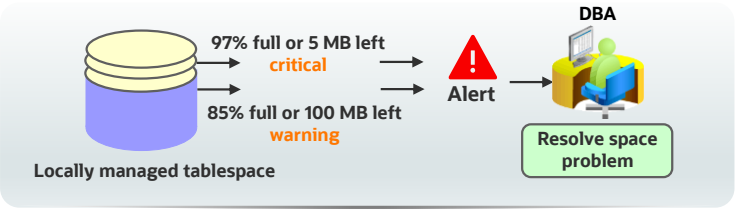
- Analyzes objects to give an estimate of space savings for different compression methods
- Helps in deciding the correct compression level for an application
- Recommends various strategies for compression
  - Picks the right compression algorithm for a particular data set
  - Sorts on a particular column for increasing the compression ratio
  - Presents tradeoffs between different compression algorithms

Compression Advisor analyzes database objects and determines the expected compression ratios that can be achieved for each compression level. It helps you determine the proper compression levels for your application and recommends various strategies for compression.

A compression advisor, provided by the `DBMS_COMPRESSION` package, helps you determine the compression ratio that can be expected for a specified table. The advisor analyzes the objects in the database, discovers the possible compression ratios that could be achieved, and recommends optimal compression levels. In addition to the `DBMS_COMPRESSION` package, compression advisor can also be used within the existing advisor framework (with the `DBMS_ADVISOR` package).

Resolving Space Usage Issues

- Resolve space usage issues by:
  - Adding or resizing data files
  - Setting `AUTOEXTEND` to `ON`
  - Shrinking objects
  - Reducing `UNDO_RETENTION`
- Check for long-running queries in temporary tablespaces.



Tablespace thresholds are defined either as full or as available space in the tablespace. Critical and warning thresholds are the two thresholds that apply to a tablespace. The `DBMS_SERVER_ALERT` package contains procedures to set and get the threshold values. When the tablespace limits are reached, an appropriate alert is raised. The threshold is expressed in terms of a percentage of the tablespace size or in remaining bytes free. It is calculated in memory. You can have both a percentage and a byte-based threshold defined for a tablespace. Either or both of them may generate an alert.

The ideal setting for the warning threshold trigger value results in an alert that is early enough to ensure that there is enough time to resolve the problem before it becomes critical, but late enough so that you are not bothered when space is not a problem.

The alert indicates that the problem can be resolved by doing one or more of the following:

- Adding more space to the tablespace by adding a file or resizing existing files or making an existing file auto-extendable
- Freeing up space on disks that contain any auto-extendable files
- Shrinking sparse objects in the tablespace

The diagram in the slide shows that the DBA receives a critical alert when the tablespace is 97% full and a warning when the tablespace is 85% full.

Reclaiming Space by Shrinking Segments

- Shrink is an online and in-place operation.
- It is applicable only to segments residing in ASSM tablespaces.
- Candidate segment types:
  - Heap-organized tables and index-organized tables
  - Indexes
  - Partitions and subpartitions
  - Materialized views and materialized view logs

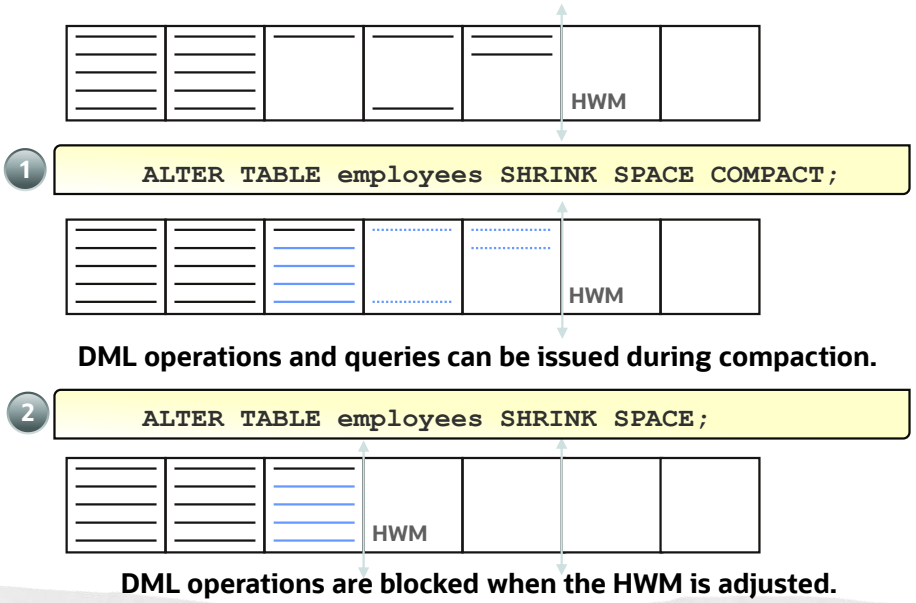
A shrink operation is an online and in-place operation because it does not need extra database space to be executed.

You cannot execute a shrink operation on segments managed by free lists. Segments in automatic segment space–managed tablespaces can be shrunk. However, the following objects stored in ASSM tablespaces cannot be shrunk:

- Tables in clusters
- Tables with `LONG` columns
- Tables with on-commit materialized views
- Tables with `ROWID`-based materialized views
- IOT mapping tables
- Tables with function-based indexes

Because a shrink operation may cause `ROWIDs` to change in heap-organized segments, you must enable row movement on the corresponding segment before executing a shrink operation on that segment. Row movement by default is disabled at segment level.

Shrinking Segments

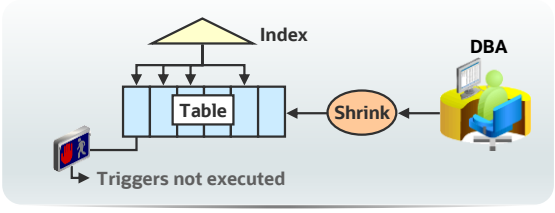


The diagram in the slide describes the two phases of a table shrink operation. Compaction is performed in the first phase. During this phase, rows are moved to the left part of the segment as much as possible. Internally, rows are moved by packets to avoid locking issues. After the rows have been moved, the second phase of the shrink operation is started. During this phase, the high-water mark (HWM) is adjusted and the unused space is released.

The `COMPACT` clause is useful if you have long-running queries that might span the shrink operation and attempt to read from blocks that have been reclaimed. When you specify the `SHRINK SPACE COMPACT` clause, the progress of the shrink operation is saved in the bitmap blocks of the corresponding segment. This means that the next time a shrink operation is executed on the same segment, the Oracle Database server remembers what has already been done. You can then reissue the `SHRINK SPACE` clause without the `COMPACT` clause during off-peak hours to complete the second phase.

Results of a Shrink Operation

- Improved performance and space utilization
- Indexes maintained
- Triggers not executed
- Number of migrated rows may be reduced
- Rebuilding secondary indexes on IOTs recommended



Shrinking a sparsely populated segment improves the performance of scan and DML operations on that segment. This is because there are fewer blocks to look at after the segment has been shrunk. This is especially true for:

- Full table scans (fewer and denser blocks)
- Better index access (fewer I/Os on range ROWID scans due to a more compact tree)

Also, by shrinking sparsely populated segments, you enhance the efficiency of space utilization inside your database because more free space is made available for objects in need.

Index dependency is taken care of during the segment shrink operation. The indexes are in a usable state after shrinking the corresponding table. Therefore, no further maintenance is needed.

The actual shrink operation is handled internally as an `INSERT/DELETE` operation. However, DML triggers are not executed because the data itself is not changed.

As a result of a segment shrink operation, it is possible that the number of migrated rows is reduced. However, you should not always depend on reducing the number of migrated rows after a segment has been shrunk. This is because a segment shrink operation may not touch all the blocks in the segment. Therefore, it is not guaranteed that all the migrated rows are handled.

Note: It is recommended to rebuild secondary indexes on an index-organized table (IOT) after a shrink operation.

Managing Resumable Space Allocation

- A resumable statement:
  - Enables you to suspend large operations instead of receiving an error
  - Gives you a chance to fix the problem while the operation is suspended, rather than starting over
  - Is suspended for the following conditions:
    - Out of space
    - Maximum extents reached
    - Space quota exceeded
  - Can be suspended and resumed multiple times

The Oracle Database server provides a means for suspending, and later resuming, the execution of large database operations in the event of space allocation failures. This enables you to take corrective action instead of the Oracle Database server returning an error to the user. After the error condition is corrected, the suspended operation automatically resumes. This feature is called “resumable space allocation.” The statements that are affected are called “resumable statements.” A statement executes in resumable mode only when the resumable statement feature has been enabled for the system or session.

Suspending a statement automatically results in suspending the transaction. Thus, all transactional resources are held through the suspension and resuming of a SQL statement. When the error condition disappears (for example, as a result of user intervention or perhaps sort space released by other queries), the suspended statement automatically resumes execution. A resumable statement is suspended when one of the following conditions occur:

- Out of space condition
- Maximum extents reached condition
- Space quota exceeded condition

A suspension timeout interval is associated with resumable statements. A resumable statement that is suspended for the timeout interval (the default is 2 hours) reactivates itself and returns the exception to the user. A resumable statement can be suspended and resumed multiple times.

Using Resumable Space Allocation

- Queries, DML operations, and certain DDL operations can be resumed if they encounter an out-of-space error.
- A resumable statement can be issued through SQL, PL/SQL, SQL*Loader, and Data Pump utilities, or Oracle Call Interface (OCI).
- A statement executes in resumable mode only if its session has been enabled by one of the following actions:
  - The `RESUMABLE_TIMEOUT` initialization parameter is set to a nonzero value.
  - An `ALTER SESSION ENABLE RESUMABLE` statement is issued.

Resumable space allocation is possible only when statements are executed within a session that has resumable mode enabled. There are two means of enabling and disabling resumable space allocation:

- Issue the `ALTER SESSION ENABLE RESUMABLE` command.
- Set the `RESUMABLE_TIMEOUT` initialization parameter to a nonzero value with an `ALTER SESSION` or `ALTER SYSTEM` statement.

When enabling resumable mode for a session or the database, you can specify a timeout period, after which a suspended statement errors out if no intervention has taken place. The `RESUMABLE_TIMEOUT` initialization parameter indicates the number of seconds before a timeout occurs. You can also specify the timeout period with the following command:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600
```

The value of `TIMEOUT` remains in effect until it is changed by another `ALTER SESSION ENABLE RESUMABLE` statement, it is changed by another means, or the session ends. The default timeout interval when using the `ENABLE RESUMABLE TIMEOUT` clause to enable resumable mode is 7,200 seconds or 2 hours.

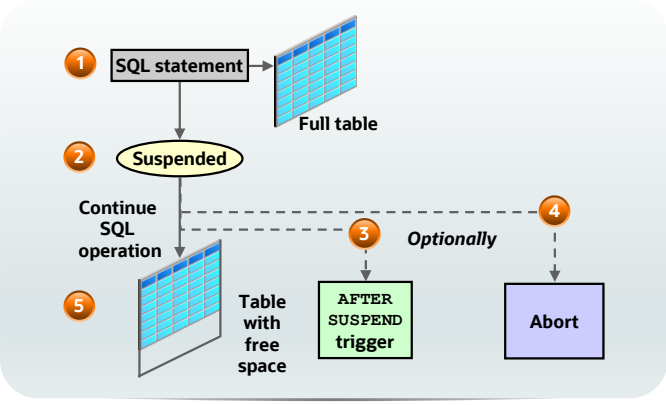
You can also give a name to resumable statements. Example:

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600 NAME 'multitab insert'
```

The name of the statement is used to identify the resumable statement in the `DBA_RESUMABLE` and `USER_RESUMABLE` views.

- To automatically configure resumable statement settings for individual sessions, you can create and register a database-level `LOGON` trigger that alters a user's session. The trigger issues commands to enable resumable statements for the session, specifies a timeout period, and associates a name with the resumable statements issued by the session.
- Because suspended statements can hold up some system resources, users must be granted the `RESUMABLE` system privilege before they are allowed to enable resumable space allocation and execute resumable statements.

Resuming Suspended Statements



When a resumable statement is suspended, the error is not raised to the client. In order for corrective action to be taken, the Oracle Database server provides alternative methods for notifying users of the error and providing information about the circumstances.

The diagram in the slide illustrates the following example:

1. An `INSERT` statement encounters an error saying the table is full.
2. The `INSERT` statement is suspended, and no error is passed to the client.
3. Optionally, an `AFTER SUSPEND` trigger is executed.
4. Optionally, the `SQLERROR` exception is activated to abort the statement.

If the statement is not aborted and free space is successfully added to the table, the `INSERT` statement resumes execution.

Possible Actions During Suspension

When a resumable statement encounters a correctable error, the system internally generates the `AFTER SUSPEND` system event. Users can register triggers for this event at both the database and schema level. If a user registers a trigger to handle this system event, the trigger is executed after a SQL statement has been suspended. SQL statements executed within an `AFTER SUSPEND` trigger are always nonresumable and autonomous. Transactions started within the trigger use the `SYSTEM` rollback segment. These conditions are imposed to overcome deadlocks and reduce the chance of the trigger experiencing the same error condition as the statement.

Within the trigger code, you can use the `USER_RESUMABLE` or `DBA_RESUMABLE` views or the `DBMS_RESUMABLE.SPACE_ERROR_INFO` function to get information about the resumable statements.

- When a resumable statement is suspended:
  - The session invoking the statement is put into a wait state. A row is inserted into `V$SESSION_WAIT` for the session with the `EVENT` column containing "statement suspended, wait error to be cleared."
  - An operation-suspended alert is issued on the object that needs additional resources for the suspended statement to complete.
- **Ending a Suspended Statement**
- When the error condition is resolved (for example, as a result of DBA intervention or perhaps sort space released by other queries), the suspended statement automatically resumes execution and the "resumable session suspended" alert is cleared.
- A suspended statement can be forced to activate the `SERVERERROR` exception by using the `DBMS_RESUMABLE.ABORT()` procedure. This procedure can be called by a DBA or by the user who issued the statement. If the suspension timeout interval associated with the resumable statement is reached, the statement aborts automatically, and an error is returned to the user.

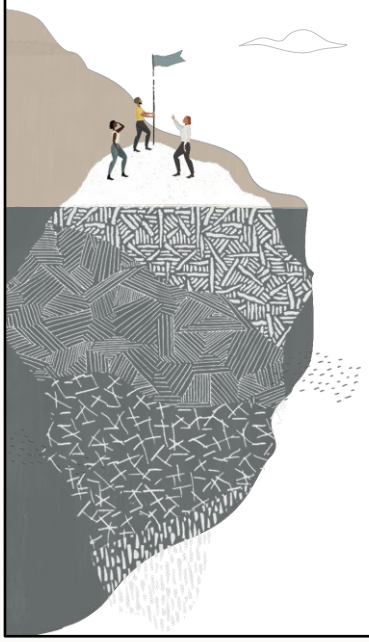
What operations are resumable?

- The following operations are resumable:
  - Queries: `SELECT` statements that run out of temporary space (for sort areas)
  - DML: `INSERT`, `UPDATE`, and `DELETE` statements
  - The following DDL statements:
    - **`CREATE TABLE ... AS SELECT`**
    - **`CREATE INDEX`**
    - **`ALTER INDEX ... REBUILD`**
    - **`ALTER TABLE ... MOVE PARTITION`**
    - **`ALTER TABLE ... SPLIT PARTITION`**
    - **`ALTER INDEX ... REBUILD PARTITION`**
    - **`ALTER INDEX ... SPLIT PARTITION`**
    - **`CREATE MATERIALIZED VIEW`**

The following operations are resumable:

- **Queries:** `SELECT` statements that run out of temporary space (for sort areas) are candidates for resumable execution. When using OCI, the `OCIStmtExecute()` and `OCIStmtFetch()` calls are candidates.
- **DML:** `INSERT`, `UPDATE`, and `DELETE` statements are candidates. The interface used to execute them does not matter; it can be OCI, SQLJ, PL/SQL, or another interface. Also, `INSERT INTO...SELECT` from external tables can be resumable.
- **DDL:** The following statements are candidates for resumable execution:
  - `CREATE TABLE ... AS SELECT`
  - `CREATE INDEX`
  - `ALTER INDEX ... REBUILD`
  - `ALTER TABLE ... MOVE PARTITION`
  - `ALTER TABLE ... SPLIT PARTITION`
  - `ALTER INDEX ... REBUILD PARTITION`
  - `ALTER INDEX ... SPLIT PARTITION`
  - `CREATE MATERIALIZED VIEW`

Summary



- Describe and use Oracle Database features that save space
- Create private temporary tables
- Save space by using compression
- Reclaim wasted space from tables and indexes by using the segment shrink functionality
- Manage resumable space allocation

ORACLE
University

Managing Undo Data

Objectives

- Explain DML and undo data generation
- Monitor and administer undo data
- Describe the difference between undo data and redo data
- Configure undo retention
- Guarantee undo retention
- Enable temporary undo

Undo Data: Overview

- Undo data is:
  - A record of the action of a transaction
  - Captured for every transaction that changes data
  - Retained at least until the transaction is ended
  - Used to support:
    - Rollback operations
    - Read-consistent queries
    - Oracle Flashback Query, Oracle Flashback Transaction, and Oracle Flashback Table
    - Recovery from failed transactions

- The Oracle Database server saves the old value (undo data) when a process changes data in a database. It stores the data as it exists before modifications. Capturing undo data enables you to roll back your uncommitted data. Undo supports read-consistent and flashback queries. Undo can also be used to "rewind" (flashback) transactions and tables.
- Read-consistent queries provide results that are consistent with the data as of the time a query started. For a read-consistent query to succeed, the original information must still exist as undo information. If the original data is no longer available, you receive a "Snapshot too old" error (`ORA-01555`). As long as the undo information is retained, the Oracle Database server can reconstruct data to satisfy read-consistent queries.
- Flashback queries purposely ask for a version of the data as it existed at some time in the past. As long as undo information for that past time still exists, flashback queries can complete successfully. Oracle Flashback Transaction uses undo to create compensating transactions, to back out a transaction and its dependent transactions. With Oracle Flashback Table, you can recover a table to a specific point in time.
- Undo data is also used to recover from failed transactions. A failed transaction occurs when a user session ends abnormally (possibly because of network errors or a failure on the client computer) before the user decides to commit or roll back the transaction. Failed transactions may also occur when the instance crashes or you issue the `SHUTDOWN ABORT` command.
- In case of a failed transaction, the safest behavior is chosen, and the Oracle Database server reverses all changes made by a user, thereby restoring the original data.

Undo information is retained for all transactions, at least until the transaction is ended by one of the following:

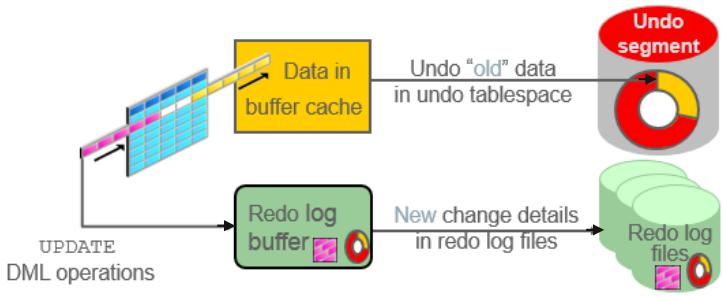
- User undoes a transaction (transaction rolls back).
- User ends a transaction (transaction commits).
- User executes a DDL statement, such as a `CREATE`, `DROP`, `RENAME`, or `ALTER` statement. If the current transaction contains any DML statements, the database server first commits the transaction and then executes and commits the DDL as a new transaction.
- User session terminates abnormally (transaction rolls back).
- User session terminates normally with an exit (transaction commits).

The amount of undo data that is retained and the time for which it is retained depend on the amount of database activity and the database configuration.

Note: Oracle Flashback Transaction leverages the online redo logs to mine the appropriate undo SQL for execution. It only uses undo as an artificial time boundary, to determine a redo mining start time for the target transaction, if a transaction start time is not supplied in the flashback transaction invocation.

Transactions and Undo Data

- Each transaction is assigned to only one undo segment.
- An undo segment can service more than one transaction at a time.



- When a transaction starts, it is assigned to an undo segment. Throughout the life of the transaction, when data is changed, the original (before the change) values are copied into the undo segment. You can see which transactions are assigned to which undo segments by checking the `V$TRANSACTION` dynamic performance view.
- Undo segments are specialized segments that are automatically created by the database server as needed to support transactions. Like all segments, undo segments are made up of extents, which, in turn, consist of data blocks. Undo segments automatically grow and shrink as needed, acting as a circular storage buffer for their assigned transactions.
- Transactions fill extents in their undo segments until a transaction is completed or all space is consumed. If an extent fills up and more space is needed, the transaction acquires that space from the next extent in the segment. After all extents have been consumed, the transaction either wraps around back into the first extent or requests a new extent to be allocated to the undo segment.

  Note: Parallel DML and DDL operations can cause multiple coordinated transactions, each of which uses its own undo segment. To learn more about parallel DML execution, see *Oracle Database Administrator's Guide*.

Storing Undo Information

- Undo information is stored in undo segments, which are stored in an undo tablespace.
- Undo tablespaces:
  - Are used only for undo segments
  - Have special recovery considerations
  - May be associated with only a single instance
  - Require that only one of them be the current writable undo tablespace for a given instance at any given time

Undo segments can exist only in a specialized form of tablespace called an undo tablespace. You cannot create other segment types, such as tables, in the undo tablespace.

The Database Configuration Assistant (DBCA) automatically creates a smallfile undo tablespace. You can also create a bigfile undo tablespace. However, in a high-volume online transaction processing (OLTP) environment with many short concurrent transactions, contention could occur on the file header. An undo tablespace, stored in multiple data files, can resolve this potential issue.

Although a database may have many undo tablespaces, only one of them at a time can be designated as the current undo tablespace for any instance in the database.

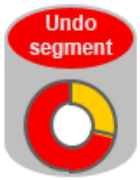
Undo segments are automatically created and always owned by `SYS`. Because the undo segments act as a circular buffer, each segment has a minimum of two extents. The default maximum number of extents depends on the database block size but is very high (32,765 for an 8 KB block size).

Undo tablespaces are permanent, locally managed tablespaces with automatic extent allocation. They are automatically managed by the database.

Because undo data is required to recover from failed transactions (such as those that may occur when an instance crashes), undo tablespaces can be recovered only while the instance is in the `MOUNT` state.

Comparing Undo Data and Redo Data

	Undo	Redo
Record of	How to undo a change	How to reproduce a change
Used for	Rollback, read consistency, flashback	Rolling forward of database changes
Stored in	Undo segments	Redo log files



Undo data and redo data seem similar at first, but they serve different purposes. Undo data is needed if there is a need to undo a change, and this occurs for read consistency and rollback. Redo data is needed if there is a need to perform the changes again, in cases where they are lost for some reason. Undo block changes are also written to the redo log.

The process of committing entails a verification that the changes in the transaction have been written to the redo log file, which is persistent storage on the disk, as opposed to memory. In addition, the redo log file is typically multiplexed. As a result, there are multiple copies of the redo data on the disk. Although the changes may not yet have been written to the data files where the table's blocks are actually stored, writing to the persistent redo log is enough to guarantee consistency of the database.

Assume that a power outage occurs just before committed changes have been reflected in the data files. This situation does not cause a problem because the transaction has been committed. When the system starts up again, it is able to roll forward any redo records that are not yet reflected in data files at the time of the outage.

Managing Undo

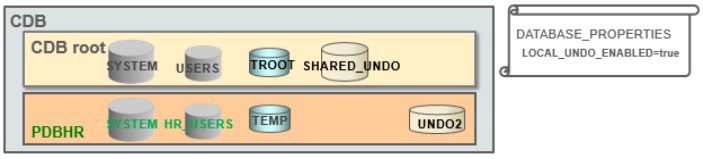
- Automatic undo management:
  - Fully automated management of undo data and space in a dedicated undo tablespace
  - For all sessions
  - Self-tuning in `AUTOEXTEND` tablespaces to satisfy long-running queries
  - Self-tuning in fixed-size tablespaces for best retention
- DBA tasks in support of Flashback operations:
  - Configuring undo retention
  - Changing the undo tablespace to a fixed size
  - Avoiding space and “snapshot too old” errors

The Oracle Database server provides automatic undo management, which is a fully automated mechanism for managing undo information and space in a dedicated undo tablespace for all sessions. The system automatically tunes itself to provide the best possible retention of undo information. More precisely, the undo retention period for auto-extending tablespaces is tuned to be slightly longer than the longest-running active query. For fixed-size undo tablespaces, the database dynamically tunes for best possible retention.

Although, by default, the Oracle Database server manages undo data and space automatically, you may need to perform some tasks if your database is using Flashback operations. The administration of undo should prevent space errors, the use of too much space, and “snapshot too old” errors.

Comparing SHARED Undo Mode and LOCAL Undo Mode

- There are two undo modes in the multitenant architecture: SHARED and LOCAL.
  - There is only one SHARED undo tablespace (in CDB root).
  - There can be a LOCAL undo tablespace in each PDB.



- When is LOCAL undo mode required?
  - Hot cloning
  - Near-zero downtime PDB relocation

```
SQL> STARTUP UPGRADE;
SQL> ALTER DATABASE LOCAL UNDO ON;
```

Using the LOCAL undo mode is required when cloning a PDB in hot mode, performing a near-zero downtime PDB relocation, refreshing PDBs, or using proxy PDBs.

You can set a CDB in LOCAL undo mode either at CDB creation or by altering the CDB property.

When the database property LOCAL_UNDO_ENABLED is FALSE, which is the default, there is only one undo tablespace that is created in the CDB root, and that is shared by all containers.

When LOCAL_UNDO_ENABLED is TRUE, every container in the CDB uses LOCAL undo, and each PDB must have its own LOCAL undo tablespace. To maintain ease of management and provisioning, undo tablespace creation happens automatically and does not require any action from the user. When a PDB is opened and an undo tablespace is not available, it is automatically created.

Configuring Undo Retention

- UNDO_RETENTION specifies (in seconds) how long already committed undo information is to be retained.
- Set this parameter when:
  - The undo tablespace has the AUTOEXTEND option enabled
  - You want to set undo retention for LOBs
  - You want to guarantee retention

- The UNDO_RETENTION initialization parameter specifies (in seconds) the low threshold value of undo retention. Set the minimum undo retention period for the auto-extending undo tablespace to be as long as the longest expected Flashback operation. For auto-extending undo tablespaces, the system retains undo for at least the time specified in this parameter and automatically tunes the undo retention period to meet the undo requirements of the queries. But this autotuned retention period may be insufficient for your Flashback operations.
- For fixed-size undo tablespaces, the system automatically tunes for the best possible undo retention period on the basis of undo tablespace size and usage history; it ignores UNDO_RETENTION unless retention guarantee is enabled. So, for automatic undo management, the UNDO_RETENTION setting is used for the three cases listed in the slide. In cases other than these three, this parameter is ignored.

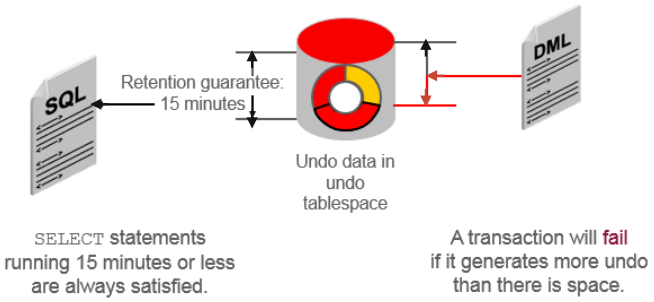
Categories of Undo

Category	Description
Active: Uncommitted undo information	Supports an active transaction and is never overwritten
Unexpired: Committed undo information	Is required to meet the undo retention interval
Expired: Expired undo information	Overwritten when space is required for an active transaction

- Undo information is divided into three categories:
  - **Uncommitted undo information (Active):** Supports a currently running transaction and is required if a user wants to roll back or if the transaction has failed. Uncommitted undo information is never overwritten.
  - **Committed undo information (Unexpired):** Is no longer needed to support a running transaction but is still needed to meet the undo retention interval. It is also known as "unexpired" undo information. Committed undo information is retained, when possible, without causing an active transaction to fail because of lack of space.
  - **Expired undo information (Expired):** Is no longer needed to support a running transaction. Expired undo information is overwritten when space is required by an active transaction.

Guaranteeing Undo Retention

```
SQL> ALTER TABLESPACE undotbs1 RETENTION GUARANTEE;
```



This example is based on an `UNDO_RETENTION` setting of 900 seconds (15 minutes).

- The default undo behavior is to overwrite the undo information of committed transactions that has not yet expired rather than to allow an active transaction to fail because of lack of undo space.
- This behavior can be changed by guaranteeing retention. With guaranteed retention, undo retention settings are enforced even if they cause transactions to fail.
- `RETENTION GUARANTEE` is a tablespace attribute rather than an initialization parameter. This attribute can be changed only with SQL command-line statements.
- The syntax to change an undo tablespace to guarantee retention is:

  ```
  SQL> ALTER TABLESPACE undotbs1 RETENTION GUARANTEE;
  ```

- To return a guaranteed undo tablespace to its normal setting, use the following command:

  ```
  SQL> ALTER TABLESPACE undotbs1 RETENTION NOGUARANTEE;
  ```

- The retention guarantee applies only to undo tablespaces. Attempts to set it on a non-undo tablespace result in the following error:

  ```
  SQL> ALTER TABLESPACE example RETENTION GUARANTEE;
  ERROR at line 1:
  ORA-30044: 'Retention' can only be specified for undo tablespace.
  ```

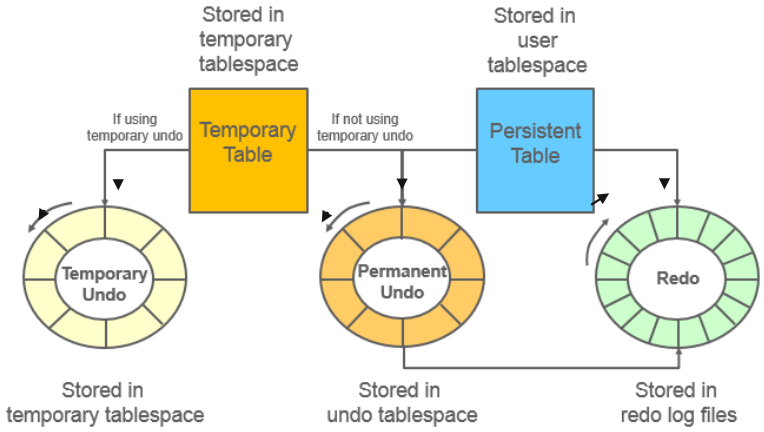
Changing an Undo Tablespace to a Fixed Size

- Rationale:
  - Supporting Flashback operations
  - Limiting tablespace growth
- Steps:
  - Run the regular workload.
  - The self-tuning mechanism establishes the minimum required size.
  - (Optional) Use the Enterprise Manager Cloud Control Undo Advisor, which calculates the required size for future growth.
  - (Optional) Change the undo tablespace to a fixed size.

- You might have two reasons for changing the undo tablespace to a fixed size: to support Flashback operations (where you expect future use of the undo) or to prevent the tablespace from growing too large.
- If you decide to change the undo tablespace to a fixed size, you must choose a large enough size to avoid the following two errors:
  - DML failures (because there is not enough space to create the undo for new transactions)
  - "Snapshot too old" errors (because there was insufficient undo data for read consistency)
- Oracle recommends that you run a regular, full workload allowing the undo tablespace to grow to its minimum required size. The automatically gathered statistics include the duration of the longest-running query and the undo generation rate. Computing the minimum undo tablespace size based on these statistics is advisable for a system without Flashback operations and for a system for which you do not expect longer-running queries in the future.
- You can use the Enterprise Manager Cloud Control Undo Advisor to enter your desired duration for the undo period for longer-running queries and flashback.

  Note: For fixed-size undo tablespaces, the system automatically tunes for the maximum possible undo retention period, based on undo tablespace size and usage history, and ignores `UNDO_RETENTION` unless retention guarantee is enabled.

Temporary Undo: Overview



Temporary tables are widely used as scratch areas for staging intermediate results. This is because changing those tables is much faster than with non-temporary tables. The performance gain is mainly because no redo entries are directly generated for changes on temporary tables. However, the undo for operations on temporary tables (and indexes) is still logged to the redo log.

Undo for temporary tables is useful for consistent reads and transaction rollbacks during the life of that temporary object. Beyond this scope, the undo is superfluous. Therefore, it need not be persisted in the redo stream. For instance, transaction recovery just discards undo for temporary objects.

Undo generated by temporary tables' transactions can be stored in a separate undo stream directly in the temporary tablespace to avoid that undo being logged in the redo stream. This mode is called temporary undo.

Note: A temporary undo segment is session private. It stores undo for the changes to temporary tables (temporary objects in general) belonging to the corresponding session.

Temporary Undo Benefits

- Reduces the amount of undo stored in the undo tablespaces
- Reduces the amount of redo data written to the redo log
- Enables DML operations on temporary tables in a physical standby database with the Oracle Active Data Guard option

Enabling temporary undo provides the following benefits:

- Temporary undo reduces the amount of undo stored in the undo tablespaces. Less undo in the undo tablespaces can result in more realistic undo retention period requirements for undo records.
- Performance is improved because less data is written to the redo log, and components that parse redo log records, such as LogMiner, perform better because there is less redo data to parse.
- Temporary undo enables data manipulation language (DML) operations on temporary tables in a physical standby database with the Oracle Active Data Guard option. However, data definition language (DDL) operations that create temporary tables must be issued on the primary database.

Enabling Temporary Undo

- Enable temporary undo for a session:

```
SQL> ALTER SESSION SET temp_undo_enabled = true;
```

- Enable temporary undo for the database instance:

```
SQL> ALTER SYSTEM SET temp_undo_enabled = true;
```

- Temporary undo mode is selected when a session first uses a temporary object.

- You can enable temporary undo for a specific session or for the entire database. When you enable temporary undo for a session using an `ALTER SESSION` statement, the session creates temporary undo without affecting other sessions. When you enable temporary undo for the system using an `ALTER SYSTEM` statement, all existing sessions and new sessions create temporary undo.
- When a session uses temporary objects for the first time, the current value of the `TEMP_UNDO_ENABLED` initialization parameter is set for the rest of the session. Therefore, if temporary undo is enabled for a session and the session uses temporary objects, then temporary undo cannot be disabled for the session. Similarly, if temporary undo is disabled for a session and the session uses temporary objects, then temporary undo cannot be enabled for the session.

Note: Temporary undo is enabled by default for a physical standby database with the Oracle Active Data Guard option. The `TEMP_UNDO_ENABLED` initialization parameter has no effect on a physical standby database with the Active Data Guard option because of the default setting.

Monitoring Temporary Undo

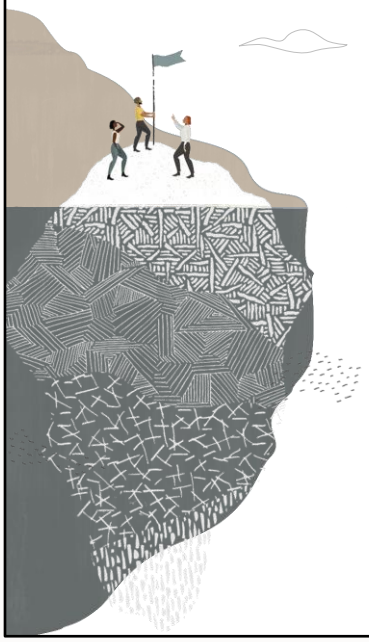
```
SQL> SELECT to_char(BEGIN_TIME,'dd/mm/yy hh24:mi:ss') "BEGIN TIME",
  2  txncount "TXNCNT", maxconcurrency, undoblkcnt, uscount "USCNT",
  3  nospaceerrcnt "NOSPEERRCNT"
  4  FROM  v$tempundostat;

BEGIN TIME        TXNCNT MAXCONCURRENCY UNDOBLKCNT USCNT NOSPEERRCNT
----------------- ------ -------------- ---------- ----- -----------
…
19/08/12 22:19:44      0              0          0     0           0
19/08/12 22:09:44      0              0          0     0           0
…
19/08/12 13:09:44      0              0          0     0           0
19/08/12 12:59:44      3              1         24     1           0
576 rows selected.
SQL>
```

- V$TEMPUNDOSTAT shows various statistics related to the temporary undo log for this database instance. It displays a histogram of statistical data to show how the system is working. Each row in the view keeps statistics collected in the instance for a 10-minute interval. The rows are in descending order of the BEGIN_TIME column value. This view contains a total of 576 rows, spanning a four-day cycle. This view is similar to the V$UNDOSTAT view.
- The example shows you some of the important columns of the V$TEMPUNDOSTAT view:
  - BEGIN_TIME: The beginning of the time interval
  - TXNCOUNT: The total number of transactions that have bound to temp undo segment within the corresponding time interval
  - MAXCONCURRENCY: The highest number of transactions executed concurrently, which modified temporary objects within the corresponding time interval
  - UNDOBLKCNT: The total number of temporary undo blocks consumed during the corresponding time interval
  - USCOUNT: The temp undo segments created during the corresponding time interval
  - NOSPACEERRCNT: The total number of times the “no space left for temporary undo” error was raised during the corresponding time interval

Note: For more information on V$TEMPUNDOSTAT, refer to *Oracle Database Reference Guide*.

Summary

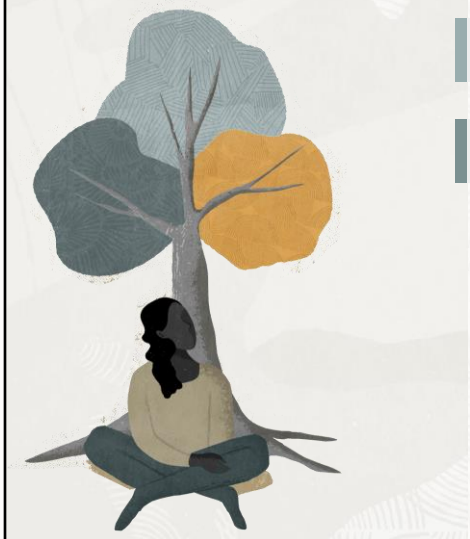


- Explain DML and undo data generation
- Monitor and administer undo data
- Describe the difference between undo data and redo data
- Configure undo retention
- Guarantee undo retention
- Enable temporary undo

ORACLE
University

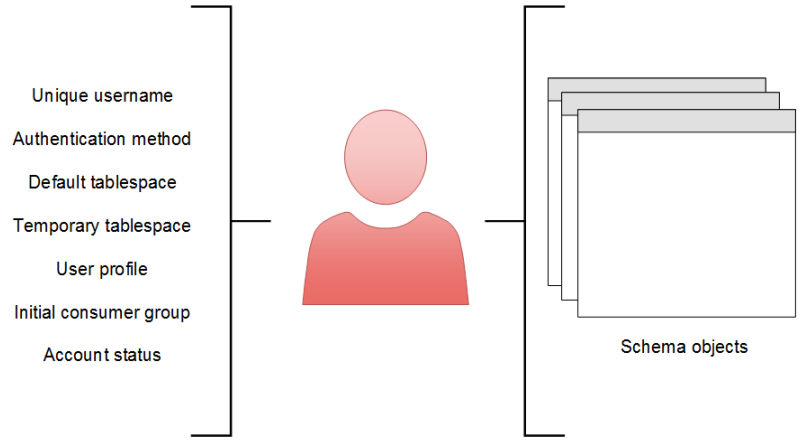
Creating and Managing User Accounts

Objectives



- Create database users
- Explain the various authentication options for users
- Assign quota to users

Database User Accounts



To access the database, a user must specify a valid database user account and successfully authenticate as required by that user account. Each database user has a unique database account. Oracle recommends this to avoid potential security holes and provide meaningful data for certain audit activities. However, users may sometimes share a common database account. In these rare cases, the operating system and applications must provide adequate security for the database.

Each user account has the following, as illustrated in the slide:

- **Unique username:** Usernames cannot exceed 30 bytes, cannot contain special characters, and must start with a letter. Usernames are not case-sensitive.
- **Authentication method:** The most common authentication method is a password.
- **Default tablespace:** This is a place where a user creates objects if the user does not specify some other tablespace.
  - Having a default tablespace does not imply that the user has the privilege of creating objects, nor does the user have a quota of space in that tablespace in which to create objects. Both of these privileges are granted separately.
  - If a user does not specify a tablespace when creating an object, the object will be created in the default tablespace assigned to the object owner. This enables you to control where the user's objects are created.
  - If an administrator does not define a default tablespace, the system-defined default permanent tablespace is used.
  - Quota for a specific tablespace is not granted through a privilege. It's done by using the `ALTER USER` command, which changes the attributes for a user. However, if a DBA grants the `UNLIMITED TABLESPACE` system privilege to a user, then that user can use all the space in any tablespace.

- **Temporary tablespace:** This is a place where temporary objects, such as sorts and temporary tables, are created on behalf of the user by the server. No quota is applied to temporary tablespaces. If an administrator does not define a temporary tablespace for a user, the system-defined temporary tablespace is used when the user creates objects.
- **User profile:** This is a set of resource and password restrictions assigned to the user.
- **Initial consumer group:** This is used by the Resource Manager.
- **Account status:** Users can access only "open" accounts. The account status may be "locked" and/or "expired."

Note: A database user is not necessarily a person. It is a common practice to create a user that owns the database objects of a particular application, such as `HR`. The database user can be a device, an application, or just a way to group database objects for security purposes. The personal identifying information of a person is not needed for a database user.

Schemas

A schema is a collection of database objects that are owned by a database user. Schema objects are the logical structures that directly refer to the database's data. Schema objects include such structures as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. In general, schema objects include everything that your application creates in the database.

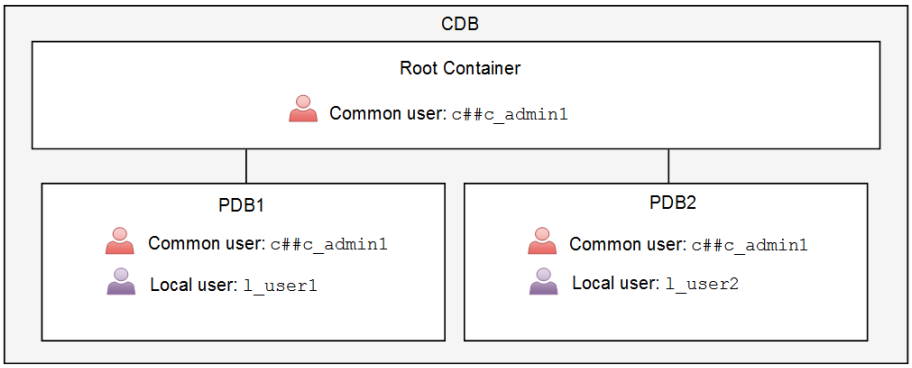
Oracle-Supplied Administrator Accounts

Account	Description
SYS	Super user. Owns the data dictionary and the Automatic Workload Repository (AWR). Used for starting up and shutting down the database instance
SYSTEM	Owns additional administrative tables and views
SYSBACKUP	Facilitates Oracle Recovery Manager (RMAN) backup and recovery operations
SYSDG	Facilitates Oracle Data Guard operations
SYSKM	Facilitates Transparent Data Encryption wallet operations
SYSRAC	For Oracle Real Application Clusters (RAC) database administration tasks
SYSMAN	For Oracle Enterprise Manager database administration tasks
DBSNMP	Used by the Management Agent component of Oracle Enterprise Manager to monitor and manage the database

The SYS and SYSTEM accounts are required accounts and cannot be deleted. You supply their passwords when you create the database instance and database in DBCA.

During installation and database creation, you can unlock and reset many of the Oracle-supplied database user accounts.

Creating Oracle Database Users in a Multitenant Environment



You can create two types of database users in a multitenant environment:

- **Common user:** The user is replicated in all existing and future containers. Oracle supplies several common user accounts for database administrators to use. A common user that you create, by default, must be given a name that starts with `C##` (for example, `c##c_admin1`). Usernames are not case sensitive. The `COMMON_USER_PREFIX` parameter specifies a prefix for common users, roles, and profiles in a container database. To create a common user, log in to the root container or application root, issue the `CREATE USER` command, and include the `CONTAINER=ALL` clause. For example:

```
SQL> CONNECT / AS SYSDBA

SQL> CREATE USER c##c_admin1 IDENTIFIED BY x CONTAINER=ALL;
```

- **Local user:** The user is created in a single PDB only. Local users cannot be created in a root container or in an application root container. A local user cannot create a common user. To create a local user, log in to the PDB where you want to create the local user and issue the `CREATE USER` command. For example:

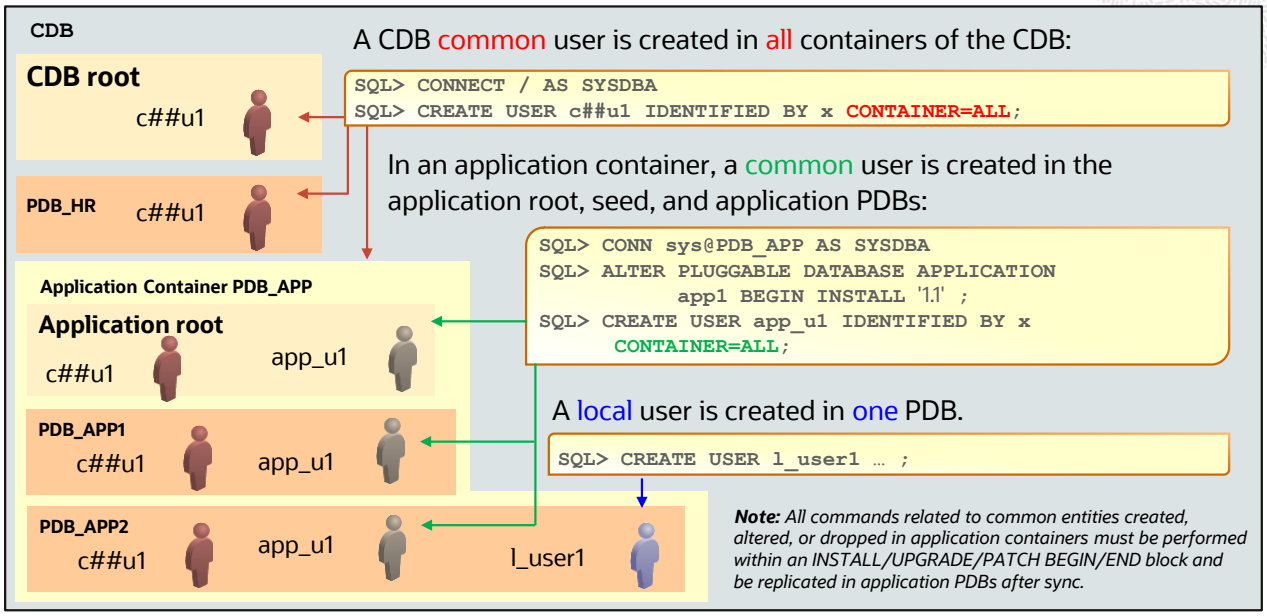
```
SQL> CONNECT SYS@PDB1 AS SYSDBA

SQL> CREATE USER l_user1 … ;
```

In the illustration in the slide, the common user `c##c_admin1` exists in the root container and all PDBs, while local user `l_user1` only exists in `PDB1` and local user `l_user2` only exists in `PDB2`.

You can use tools such as SQL*Plus and SQL Developer to create user accounts in the Oracle database.

Creating Common Users in the CDB and PDBs



A common user is a user that has the same user name and authentication credentials across multiple PDBs of the CDB or an application container, unlike a local user that exists in only one PDB.

A common user cannot have the same name as any local user across all the PDBs. A common user can be created in the CDB root or in an application root: a common user is a database user that has the same identity in the CDB root and in every existing and future PDB in the CDB or in an application root and in every existing and future application PDB in the application container. An application common user does not require a prefix like a CDB common user.

To create an application common user, you must be logged in to the application root. The application common user is replicated in all application PDBs when the application PDBs are synchronized with the application root.

A local user can be created in a specific PDB and cannot be created in the CDB root or in an application root. A local user cannot create a common user.

Note: If an application PDB is closed, the CDB common users, application common users, and local users of the application PDB are not visible because the metadata is retrieved from the PDB `SYSTEM` tablespace.

Creating Schema Only Accounts

- It ensures that a user cannot log in to the instance.
- It enforces data access through the application.
- It secures schema objects by preventing the connected schema from dropping objects.
- A schema only account cannot connect through a database link.
- Oracle-supplied schemas created with the `NO AUTHENTICATION` clause are schema only accounts.
- Administrator privileges can be granted to and revoked from schema only accounts.

```
SQL> CREATE USER schema_noauth NO AUTHENTICATION;
```

Application designers may want to create accounts that contain the application data dictionary, but are not allowed to log in to the instance. This can be used to enforce data access through the application, separation of duties at the application level, and other security mechanisms.

In addition, utility accounts can be created but remain inaccessible by denying the ability to log in except under controlled situations.

You can create a user account with the `NO AUTHENTICATION` clause to ensure that the account is not permitted to log in to the instance. Removing the password and the ability to log in essentially leaves just a schema. The schema account can be altered to allow login, but can then have the password removed. The `ALTER USER` statement can be used to disable or re-enable the login capability.

The `AUTHENTICATION_TYPE` column in the `DBA_USERS` view contains `NONE` when `NO AUTHENTICATION` is set, and `PASSWORD` when a password is set for the user account.

Most of the Oracle-supplied schemas are schema only accounts, including the Database Vault–supplied schemas such as `DVSYS` and `DVF`, and the Oracle Label Security–supplied `LBACSYS` schema. The benefit of this functionality is that administrators no longer have to periodically rotate the passwords for these Oracle Database–provided schemas. This functionality also reduces the security risk of attackers using default passwords to hack into these accounts.

Accounts with administrator privileges such as `SYSOPER` or `SYSBACKUP` can also be schema-only accounts. Schema-only accounts can be granted administrator privileges.

Authenticating Users

- Every user, including administrators, must be authenticated when connecting to a database instance.
- Authentication verifies that the user is a valid database user and establishes a trust relationship for further interactions.
- Authentication also enables accountability by making it possible to link access and actions to specific identities.

Your choice of authentication is influenced by whether you intend to administer your database locally on the same system where the database resides or whether you intend to administer many different databases from a single remote client.

Authenticating Users

- The following authentication methods are possible:
  - Password (usually for database users)
  - Operating system (OS) authentication
  - Password file (for system administrative privileged users only)
  - Strong authentication with Kerberos, SSL, or directory authentication
- A system administrative privileged user must use OS authentication, password file authentication, or strong authentication.
  - These methods can authenticate when the database is available or unavailable (not started).

Using Password Authentication

- Create each user with an associated password that must be supplied when the user attempts to establish a connection.
- When setting up a password, you can expire the password immediately, which forces the user to change the password after first logging in.
- All passwords created in Oracle Database are case-sensitive by default.
- Passwords may contain multibyte characters and are limited to 30 bytes.

Password authentication is also referred to as “authentication by the Oracle Database server.”

If you decide to set password expiration, make sure that users have the ability to change the password. Some applications do not have this functionality.

Using Password Authentication

- Passwords are always automatically and transparently encrypted by using the Advanced Encryption Standard (AES) algorithm during network (client/server and server/server) connections before sending them across the network.
- A password management policy, controlled through user profiles, can be used to:
  - Set a password expiration period
  - Grace period for changing a password, and
  - Other attributes.

Password authentication is also referred to as “authentication by the Oracle Database server.”

If you decide to set password expiration, make sure that users have the ability to change the password. Some applications do not have this functionality.

Using Password File Authentication

- You can use password file authentication for an Oracle database instance and for an Oracle Automatic Storage Management (Oracle ASM) instance.
- A password file stores database usernames and case-sensitive passwords for administrator users (common and local administrators).
- To prepare for password file authentication, you must:
  - Create the password file. DBCA creates a password file during execution.
  - Set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter
  - Grant system administrative privileges (for example, `GRANT SYSDBA TO mydba`)

For more information, see the following sources in *Oracle Database Administrator's Guide:*

- Preparing to Use Password File Authentication
- Connecting Using Password File Authentication

On UNIX and Linux, the password file is called `orapwORACLE_SID` and is stored in `$ORACLE_HOME/dbs`. On Windows, the file is called `PWDORACLE_SID.ora` and is stored in `$ORACLE_HOME\database`.

If your concern is that the password file might be vulnerable or that the maintenance of many password files is a burden, strong authentication can be implemented.

You can query `V$PWFILE_USERS` to view information in the password file.

Using OS Authentication

- Oracle Universal Installer creates operating system groups, assigns them specific names, and maps each group to a specific system privilege.
  - Example: Members of the `dba` group are granted `SYSDBA`
- As a group member, you can be authenticated, enabled as an administrative user, and connected to a local database:

```
SQL> CONNECT / AS SYSDBA
SQL> CONNECT / AS SYSOPER
SQL> CONNECT / AS SYSBACKUP
SQL> CONNECT / AS SYSDG
SQL> CONNECT / AS SYSKM
SQL> CONNECT / AS SYSRAC
```

- If you are not a member of one of these OS groups, you will not be able to connect as an administrative user via OS authentication.

Oracle Universal Installer creates operating system groups, assigns them specific names, and maps each group to a specific system privilege. The table in the next slide shows this mapping for a UNIX or Linux environment. Membership in one of these operating system groups enables a database administrator to authenticate to the database instance through the operating system rather than with a database username and password. This is known as operating system authentication.

The special system privileges are not exercised unless you include them in your `CONNECT` clause. For example, assume that the `HR` user is granted the `SYSDBA` privilege and connects with that privilege. Notice that the current user becomes `SYS`:

```
SQL> CONNECT hr@PDB1 AS SYSDBA
Enter password: ******
Connected.
SQL> SHOW USER
USER is "SYS"
```

However, if the `HR` user logs in to `PDB1` without including the `AS SYSDBA` clause, the current user is `HR` and the user does not have the `SYSDBA` privilege.

```
SQL> CONNECT hr@PDB1
Enter password: ******
Connected.
SQL> SHOW USER
USER is "HR"
```

If your operating system permits, you can have it authenticate users. They will not need to provide a username or password when connecting to the database instance.

If you use operating system authentication, set the `OS_AUTHENT_PREFIX` initialization parameter and use this prefix in Oracle usernames. The `OS_AUTHENT_PREFIX` parameter defines a prefix that the Oracle database adds to the beginning of each user's operating system account name. The default value of this parameter is `OPS$` for backward compatibility with the previous versions of the Oracle software. The Oracle database compares the prefixed username with the Oracle usernames in the database when a user attempts to connect. For example, assume that `OS_AUTHENT_PREFIX` is set as follows:

```
OS_AUTHENT_PREFIX=OPS$
```

If a user with an operating system account named `tsmith` needs to connect to an Oracle database and be authenticated by the operating system, the Oracle database checks whether there is a corresponding database user `OPS$tsmith` and, if so, allows the user to connect. All references to a user who is authenticated by the operating system must include the prefix, as seen in `OPS$tsmith`. The text of the `OS_AUTHENT_PREFIX` initialization parameter is case-sensitive on some operating systems.

OS Authentication for Privileged Users

OS Group	UNIX or Linux User Group	Special System Privilege Granted to Members
Oracle Software Group (top level group)	`oinstall`	Allowed to create and delete database files on the OS. All database administrators belong to this group.
Database Administrator Group (`OSDBA`)	`dba`	`SYSDBA` (Connects you as the `SYS` user)
Database Operator Group (`OSOPER`) – optional	`oper`	`SYSOPER` (Connects you as the `PUBLIC` user)
Database Backup and Recovery Group (`OSBACKUPDBA`)	`backupdba`	`SYSBACKUP`
Data Guard Administrative Group (`OSDGDBA`)	`dgdba`	`SYSDG`
Encryption Key Management Administrative Group (`OSKMDBA`)	`kmdba`	`SYSKM`
Real Application Cluster Administrative Group (`OSRACDBA`)	`rac`	`SYSRAC`

If you are not a member of one of the OS groups listed in the slide, you will not be able to connect as an administrative user via OS authentication. That is, `CONNECT / AS SYSDBA` will fail. However, you can still connect using other authentication methods (for example, network, password, or directory-based authentication).

Assigning Quotas

- A quota is a space allowance in a given tablespace.
- By default, a user has no quota on any of the tablespaces.
- Database accounts that need quota are those that own database objects
  - Example: Accounts for applications
- Only those activities that use space in a tablespace count against quota.

A *quota* is a space allowance in a given tablespace. By default, a user has no quota on any of the tablespaces.

You must not provide a quota to users on the `SYSTEM` or `SYSAUX` tablespaces. Typically, only the `SYS` and `SYSTEM` users are able to create objects in the `SYSTEM` or `SYSAUX` tablespaces.

You do not need a quota on an assigned temporary tablespace or any undo tablespaces. You do not need to have a quota to insert, update, and delete data in an Oracle database. The only users that need quota are the accounts that own the database objects. It is typical when installing application code that the installer creates database accounts to own the objects. Only these accounts need quotas. Other database users can be granted permission to use these objects without a quota.

The Oracle server checks the quota when a user creates or extends a segment.

For activities that are assigned to a user schema, only those activities that use space in a tablespace count against the quota. Activities that do not use space in the assigned tablespace do not affect the quota (such as creating views or using temporary tablespaces).

The quota is replenished when objects owned by the user are dropped with the `PURGE` clause or when the objects owned by the user in the recycle bin are purged.

Assigning Quotas

- Oracle server checks quota when you create or extend a segment.
- Activities that don't use space don't impact quota
  - Example: `CREATE VIEW`
- You can be granted permission to use objects without needing any quota.
- Quota is not needed for assigned temporary tablespaces or undo tablespaces.
- A user's quota is replenished when objects are dropped with the `PURGE` clause or when objects in the recycle bin are purged.

A *quota* is a space allowance in a given tablespace. By default, a user has no quota on any of the tablespaces.

You must not provide a quota to users on the `SYSTEM` or `SYSAUX` tablespaces. Typically, only the `SYS` and `SYSTEM` users are able to create objects in the `SYSTEM` or `SYSAUX` tablespaces.

You do not need a quota on an assigned temporary tablespace or any undo tablespaces. You do not need to have a quota to insert, update, and delete data in an Oracle database. The only users that need quota are the accounts that own the database objects. It is typical when installing application code that the installer creates database accounts to own the objects. Only these accounts need quotas. Other database users can be granted permission to use these objects without a quota.

The Oracle server checks the quota when a user creates or extends a segment.

For activities that are assigned to a user schema, only those activities that use space in a tablespace count against the quota. Activities that do not use space in the assigned tablespace do not affect the quota (such as creating views or using temporary tablespaces).

The quota is replenished when objects owned by the user are dropped with the `PURGE` clause or when the objects owned by the user in the recycle bin are purged.

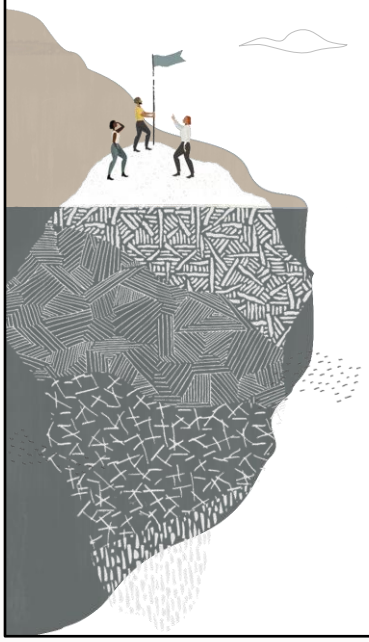
Assigning Quotas

- Options to provide quota for a user on a tablespace:
  - `UNLIMITED`
  - Value
  - `UNLIMITED TABLESPACE` system privilege

You have three options for providing a quota for a user on a tablespace:

- `UNLIMITED`: Allows the user to use as much space as is available in the tablespace
- Value: Number of kilobytes or megabytes that the user can use. This does not guarantee that the space is set aside for the user. This value can be larger or smaller than the current space that is available in the tablespace.
- `UNLIMITED TABLESPACE` system privilege: Overrides all individual tablespace quotas and gives the user unlimited quota on all tablespaces, including `SYSTEM` and `SYSAUX`. This privilege must be granted with caution.

Summary



- Create database users
- Explain the various authentication options for users
- Assign quota to users

ORACLE
University

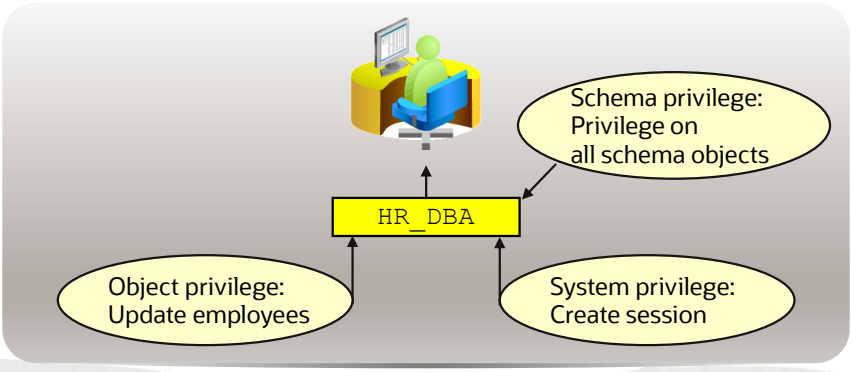
Configuring Privilege and Role Authorization

Objectives

- Grant system, schema, and object privileges to database users, commonly and locally
- Create roles
- Grant roles to users and other roles, commonly and locally
- Revoke privileges and roles from users and other roles

Privileges

- **Object:** Enables users to access and manipulate a specific object
- **Schema:** Enables certain system privileges to be granted on all objects of a schema
- **System:** Enables users to perform particular actions in the database



A **privilege** is a right to execute a particular type of SQL statement or access another user's object.

Privileges are divided into three categories:

- **Object privileges:** Object privileges allow a user to perform a particular action on a specific object, such as a table, view, sequence, procedure, function, or package. Without specific permission, users can access only their own objects. Object privileges can be granted by the owner of an object, by the administrator, or by someone who has been explicitly given permission to grant privileges on the object.
- **Schema privileges:** This feature allows certain system privileges to be granted on all objects of a schema. When such a schema-level privilege is granted, the grantee will have the privilege on all the objects in the schema on which the grant has been made. These privileges will also be applicable on all the objects that will be created in the future in the schema on which the grant has been made. Schema-level privilege requires a separate level of validation between Object and System privileges, therefore, it could slightly effect SQL performance.
- **System privileges:** Each system privilege allows a user to perform a particular database operation or class of database operations. For example, the privilege to create tablespaces is a system privilege. System privileges can be granted by the administrator or by someone who has been given explicit permission to administer the privilege. There are more than 170 distinct system privileges. Many system privileges contain the `ANY` clause.

System Privileges

- Each system privilege allows a user to perform a particular database operation or class of database operations.
- Administrators have special system privileges.
- A system privilege with the `ANY` clause means the privilege applies to all schemas except `SYS`, not just your own.
- If you grant a system privilege with the `ADMIN OPTION` enabled, you enable the grantee to administer the system privilege and grant it to other users.

Each system privilege allows a user to perform a particular database operation or class of database operations. System privileges can be granted by the administrator or by someone who has been given explicit permission to administer the privilege. You can administer system privileges when you create a user or at a later time. Carefully consider security requirements before granting system permissions.

There are more than 250 distinct system privileges. A few are listed here:

- `CREATE SESSION`: Enables a user to connect to a database instance
- `DROP ANY OBJECT`
- `CREATE TABLESPACE`
- `DROP TABLESPACE`
- `ALTER TABLESPACE`
- `CREATE LIBRARY`
- `CREATE ANY DIRECTORY`
- `GRANT ANY OBJECT PRIVILEGE`
- `ALTER DATABASE`
- `ALTER SYSTEM`

ANY Clause

Many system privileges contain an ANY clause, which means the privilege applies to all schemas, not just your own. For example, the SELECT ANY TABLE system privilege allows you to retrieve data from all tables and views, including those from schemas owned by other users. The SYS user and users with the DBA role are granted all the ANY privileges; therefore, they can do anything to any data object. You can control the scope of all system privileges, including those with the ANY clause, by using Oracle Database Vault.

ADMIN OPTION

If you grant a system privilege with ADMIN OPTION enabled, you enable the grantee to administer the system privilege and grant it to other users. The SQL syntax for granting system privileges is:

```
SQL> GRANT <system_privilege> TO <grantee clause> [WITH ADMIN OPTION]
```

System Privileges for Administrators

Privilege	Description
SYSDBA	Perform all administrative tasks in the database, including create and drop a database, open and mount a database, start up and shut down an Oracle database, create an SPFILE, put a database in or remove a database from ARCHIVELOG mode, perform incomplete recovery operations, patch, and migrate. This privilege enables you to connect as the SYS user.
SYSOPER	Perform similar administration tasks as the SYSDBA privilege, but without the ability to look at user data. For example, you can start up and shut down the database, create an SPFILE, and perform complete recovery operations (not incomplete recovery operations).
SYSASM	Start up, shut down, and administer an Automatic Storage Management instance.
SYSBACKUP	Perform backup and recovery operations by using RMAN or SQL*Plus.
SYSDG	Perform Data Guard operations by using Data Guard Broker or the DGMGRL command-line interface.
SYSKM	Manage Transparent Data Encryption wallet operations.
SYSRAC	Perform day-to-day administration tasks on an Oracle Real Application Clusters (RAC) cluster.

There are seven special system privileges that are usually granted only to administrators. Anyone who is granted one of these privileges is referred to as a system administrative privileged user (privileged user, for short).

Only users who are granted the SYSDBA, SYSOPER, SYSASM, and SYSRAC privileges are allowed to start up and shut down the Oracle database.

The SYSBACKUP, SYSDG, and SYSKM privileges enable you to connect to the database even if the database is not open.

Users with explicit object privileges or those who connect with administrative privileges (SYSDBA) can access objects in the SYS schema. You can grant the SELECT ANY DICTIONARY system privilege to users who require access to tables created in the SYS schema. This system privilege allows query access to any object in the SYS schema, including tables created in that schema.

Schema-Level Privileges

Schema-Level Privilege Grants:

- This feature allows certain system privileges to be granted on a schema.
  - When such a schema-level privilege is granted, the grantee will have the privilege on all the objects in the schema on which the grant has been made.
  - These privileges will also be applicable on all the objects that will be created in the future in the schema on which the grant has been made.

Schema-Level Privilege Grants

This feature allows certain system privileges to be granted on a schema. When such a schema-level privilege is granted, the grantee will have the privilege on all the objects in the schema on which the grant has been made. These privileges will also be applicable on all the objects which will be created in the future in the schema on which the grant has been made.

Of all the system privileges supported by Oracle database, the system privileges that are not associated with schema qualified objects will not be considered for schema-level privileges. For example, schema objects like directory and dictionary are always owned by SYS. Hence, further scoping of these relevant system privileges to schema-level privileges is not required. The following is the list of commonly used system privileges that will not be part of a schema-level privilege:

- Administrative Privileges: sysdba, sysoper, sysasm, sysbackup, sysdg, syskm
- System: alter database, alter system, audit system, alter resource cost
- Session: create session, alter session, restrict session
- Resumable Space Allocation: resumable
- Tablespace: create tablespace, alter tablespace, manage tablespace, drop tablespace, unlimited tablespace
- Rollback Segment: create rollback segment, alter rollback segment, drop rollback segment
- Transaction: force transaction, force any transaction
- Resource Management: administrate resource manager
- User: create user, become user, alter user, drop user

- Role: create role, drop any role, grant any role, alter any role
- Profile: create profile, alter profile, drop profile
- Public Synonym: create public synonym, drop public synonym
- Database Link: create database link, create public database link, drop public database link
- Directory: create any directory, drop any directory, read, write
- Pluggable Database: create pluggable database, set container
- Application Context: create any context, drop any context
- Stored Outline: create any outline, alter any outline, drop any outline
- Database Trigger: administer database trigger
- Debugging: debug connect session
- Dictionary Protection: select any dictionary, analyze any dictionary
- Export/Import: export full database, import full database
- Advisor Framework: advisor, administer sql tuning set
- Edition: create any edition, drop any edition
- Flashback: flashback archive administer, select any transaction
- Key Management: administer key management
- Logminer: Logmining
- Plan Management: administer SQL management object
- Database Change Notification: change notification
- Application Continuity: keep date time, keep sysguid
- Recycle Bin: Purge DBA_Recyclebin

Schema-Level Privileges

Schema-Level Privilege Performance:

- With the addition of a schema-level privilege, one more check is needed in between the object privilege and system privilege checks.
  - Determines if the user has the relevant privilege on the schema of the object to satisfy the requested access.
- This additional check takes negligible amount of time for statement execution and, therefore, no performance tuning is required.

Schema-Level Privilege Performance

Previously, when the privilege check happened on an object for any access, a check was made to determine if the user had object privilege on the object. If the user didn't have the object privilege, then a check was made to determine if the user had the much more powerful system privilege, which would satisfy the access requested. With the addition of schema-level privilege, one more check will be needed in between the object privilege and system privilege checks, to determine if the user has the relevant privilege on the schema of the object to satisfy the requested access. This additional check will add a negligible amount of time to statement execution and, therefore, no performance tuning will be required.

Schema-Level Privileges

Schema-Level Privilege Security:

- Schema-level privileges can be granted by the following:
  - Schema owners who will be able to grant a schema-level privilege on their own schema to any user or role
  - Users who have a schema-level privilege with `WITH ADMIN OPTION` will be able to grant that schema-level privilege to any user or role
  - Users with the `GRANT ANY SCHEMA PRIVILEGE` system privilege will be able to grant any schema-level privilege to any user or role

Schema-Level Privilege Security

Schema-level privileges will have to be granted to users with due diligence as they provide access to all the objects in the schema. Schema-level privileges can be granted by:

- Schema owners, who will be able to grant a schema-level privilege on their own schema to any user or role
- Users who have a schema-level privilege with `WITH ADMIN OPTION` will be able to grant that schema-level privilege to any user or role
- Users with the `GRANT ANY SCHEMA PRIVILEGE` system privilege will be able to grant any schema-level privilege to any user or role

Schema-Level Privileges

Granting and Revoking a Schema-Level Privilege:

- Granting a schema-level privilege:

  Grant <Schema-Privilege> on schema <Schema-Name> to user|role

  Grant ALL PRIVILEGES on schema <Schema-Name> to user|role

- Revoking a schema-level privilege:

  Revoke <Schema-Privilege> on schema <Schema-Name> from user|role

  Revoke ALL PRIVILEGES on schema <Schema-Name> from user|role

Schema-Level Privileges

Viewing Schema-Level Privileges:

- The following views can be used to display schema-level privileges:
  - DBA_SCHEMA_PRIVS
  - USER_SCHEMA_PRIVS
  - ROLE_SCHEMA_PRIVS
  - SESSION_SCHEMA_PRIVS

Four views that display schema-level privileges will be created. These are:

- **DBA_SCHEMA_PRIVS**: This view can be accessed by the DBA role and lists all the schema-level privileges granted to users or roles in the database.
- **USER_SCHEMA_PRIVS**: This view can be accessed by the current user in the session and lists all the schema-level privileges granted to the current user.
- **ROLE_SCHEMA_PRIVS**: This view can be accessed by the current user in the session and lists all the schema-level privileges granted to the enabled roles of current user.
- **SESSION_SCHEMA_PRIVS**: This view can be accessed by the current user in the session and lists all the schema-level privileges granted to the current user and the schema-level privileges granted to the enabled roles of the current user.

Viewing Privilege Analysis Results for Schema-Level Privileges:

The following views can be used to display the results of the analysis generated using the DBMS_PRIVILEGE_CAPTURE.GENERATE_RESULT procedure:

- **DBA_USED_SCHEMA_PRIVS**
- **DBA_USED_SCHEMA_PRIVS_PATH**
- **DBA_UNUSED_SCHEMA_PRIVS**
- **DBA_UNUSED_SCHEMA_PRIVS_PATH**

New Developer Role and Simplified Schema Privileges

Schema-Level Privilege Example:

```
$ sqlplus sys/WElcome123##@pdbsec as sysdba
Connected.

SQL> Grant CEATE ANY TABLE, INSERT ANY TABLE, SELECT ANY TABLE
on schema HR to app_dev;

Grant succeeded.
```

Object Privileges

- These privileges allow a user to perform a particular action on a specific object.
- Without specific permission, users can access only their own objects.
- Object privileges can be granted by:
  - The owner of an object
  - The administrator
  - Someone who has been explicitly given permission to grant privileges on the object
- The SQL syntax for granting object privileges is:

```
GRANT <object_privilege> ON <object> TO <grantee clause>
[WITH GRANT OPTION]
```

Object privileges allow a user to perform a particular action on a specific object, such as a table, view, sequence, procedure, function, or package.

Object privileges can be granted by the owner of an object, by the administrator, or by someone who has been explicitly given permission to grant privileges on the object.

Granting Privileges in a Multitenant Environment

- Commonly: Grant the user a privilege in all containers of a CDB.

```
SQL> CONNECT / AS SYSDBA
SQL> GRANT create session TO c##c_admin1 CONTAINER=ALL;
```

- Locally: Grant the user a privilege in a single PDB only.

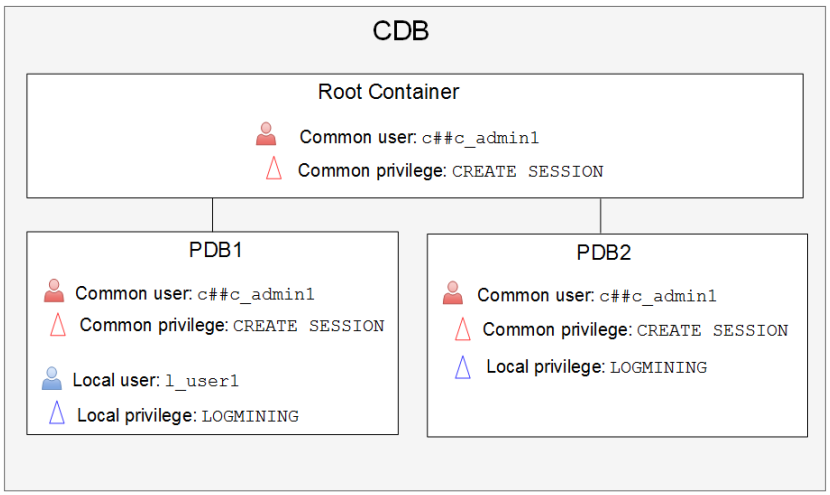
```
SQL> CONNECT SYS@PDB1 AS SYSDBA
SQL> GRANT logmining TO l_user1;
```

In a multitenant environment, you can grant a privilege to a user in two ways:

- **Commonly:** You grant the user the privilege in all containers of a CDB. In an application container, a common privilege is granted to a grantee in the application root and application PDBs. To grant a privilege commonly, you must log in to the root container or an application root container and issue the `GRANT` command with the `CONTAINER=ALL` clause.
- **Locally:** You grant the user the privilege in a single PDB only. To grant a privilege locally, log in to the PDB and issue the `GRANT` command.

To switch to a different container, a common user must have the `SET CONTAINER` privilege in the current container. Alternatively, a common user can start a new database session whose initial current container is the container the user wants, relying on the `CREATE SESSION` privilege in that PDB. Be aware that commonly granted privileges that have been made to common users may interfere with the security configured for individual PDBs.

Granting Privileges: Example



The diagram in the slide illustrates the following:

- The `c##c_admin1` common user is granted the `CREATE SESSION` privilege commonly, which applies the privilege to that user in all containers. In `PDB2`, `c##c_admin1` is also granted the `LOGMINING` privilege locally. `c##c_admin1` does not have the `LOGMINING` privilege in any other container.
- The local user, `l_user1`, exists in `PDB1` only and is granted the `LOGMINING` privilege. If the same user existed in another PDB (as a totally separate user), the `LOGMINING` privilege would not be applied.

Considerations

Step 1: Create a common user when the same user has to perform the same actions in all PDBs in the CDB. Otherwise, create the user as a local user in a PDB.

Step 2: Ask yourself, do you want the common user who exists in each PDB to have the same privileges in the PDBs?

- If yes, then you commonly grant the privileges to the common user. Connect to the CDB as a user who is privileged enough to do it and grant privilege1, privilege2, and so on to the common user by using the `CONTAINER=ALL` clause.
- If no, then you locally grant the privileges to the common user. Connect to the PDB as a user who is privileged enough to do it and grant privilege1, privilege2, and so on to the common user.

Using Roles to Manage Privileges

- Roles:
  - Used to group privileges and roles together
  - Facilitate the granting of multiple privileges or roles to users
- Benefits of roles:
  - Easier privilege management
  - Dynamic privilege management
  - Selective availability of privileges

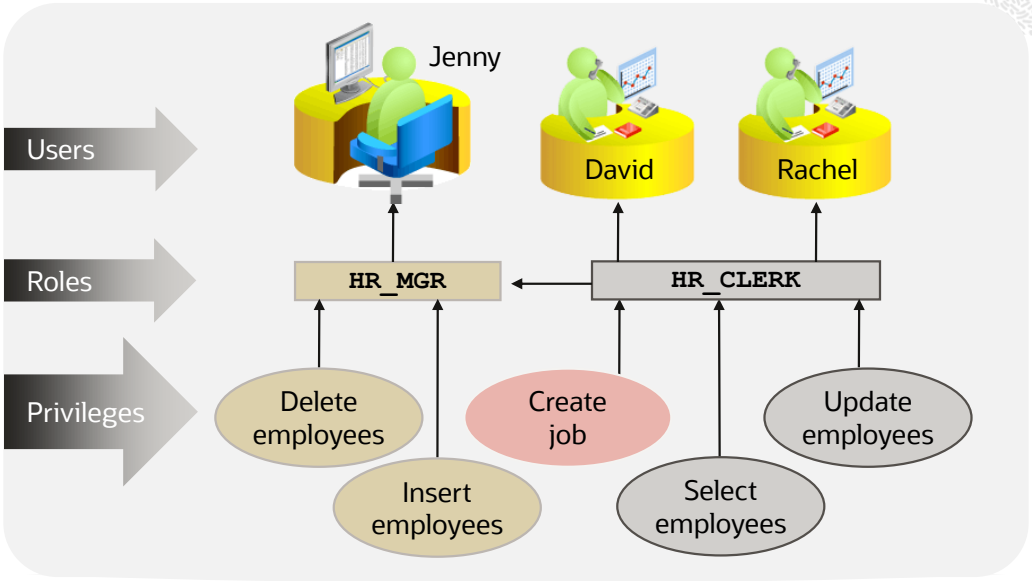
A role is a named group of related privileges that are granted to users or other roles.

You can use roles to administer database privileges. You can add privileges to a role and grant the role to a user. The user can then enable the role and exercise the privileges granted by the role. A role contains all privileges that are granted to that role and all privileges of other roles that are granted to it.

Roles provide the following benefits with respect to managing privileges:

- **Easier privilege management:** Use roles to simplify privilege management. Rather than granting the same set of privileges to several users, you can grant the privileges to a role and then grant that role to each user.
- **Dynamic privilege management:** If the privileges associated with a role are modified, all users who are granted the role acquire the modified privileges automatically and immediately.
- **Selective availability of privileges:** Roles can be enabled and disabled to turn privileges on and off temporarily. This allows the privileges of the user to be controlled in a given situation.

Assigning Privileges to Roles and Assigning Roles to Users



In most systems, it is time-consuming and error-prone to grant necessary privileges to each user individually. Oracle software provides for easy and controlled privilege management through roles. Roles are named groups of related privileges that are granted to users or other roles. Roles are designed to ease the administration of privileges in the database and, therefore, improve security.

Role Characteristics

- Privileges are granted to and revoked from roles as though the role were a user.
- Roles are granted to and revoked from users or other roles as though they were system privileges.
- A role can consist of both system and object privileges.
- A role can be enabled or disabled for each user who is granted the role.
- A role can require a password to be enabled.
- Roles are not owned by anyone, and they are not in any schema.

In the example in the slide, the `SELECT` and `UPDATE` privileges on the `employees` table *and* the `CREATE JOB` system privilege are granted to the `HR_CLERK` role. `DELETE` and `INSERT` privileges on the `employees` table *and* the `HR_CLERK` role are granted to the `HR_MGR` role.

The manager is granted the `HR_MGR` role and can now select, delete, insert, and update the `employees` table.

Oracle-Supplied Roles

Account	Description
DBA	Includes most system privileges and several other roles. Do not grant this role to nonadministrators. Users with this role can connect to the CDB or PDB only when it is open.
RESOURCE	CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE
SCHEDULER_ADMIN	CREATE ANY JOB, CREATE EXTERNAL JOB, CREATE JOB, EXECUTE ANY CLASS, EXECUTE ANY PROGRAM, MANAGE SCHEDULER
SELECT_CATALOG_ROLE	SELECT privileges on data dictionary objects
DB_DEVELOPER_ROLE	The Developer role will be granted to the application schema owner account to provide all the required privileges for designing and managing the application schema using the most complex new schema structures such as Analytical Views, Hierarchy, Attribute Dimension, etc.

The table in the slide lists some commonly used predefined roles in Oracle Database.

You must not alter the privileges granted to Oracle-supplied roles without the assistance of Oracle Support because you may inadvertently disable the needed functionality. The SYS and SYSTEM accounts have the DBA role granted to them by default.

Granting Roles in a Multitenant Environment

- Commonly: Grant the role to the user (or role) in all containers.

```
SQL> CONNECT / AS SYSDBA
SQL> GRANT <common role> TO <common user or role> CONTAINER=ALL;
```

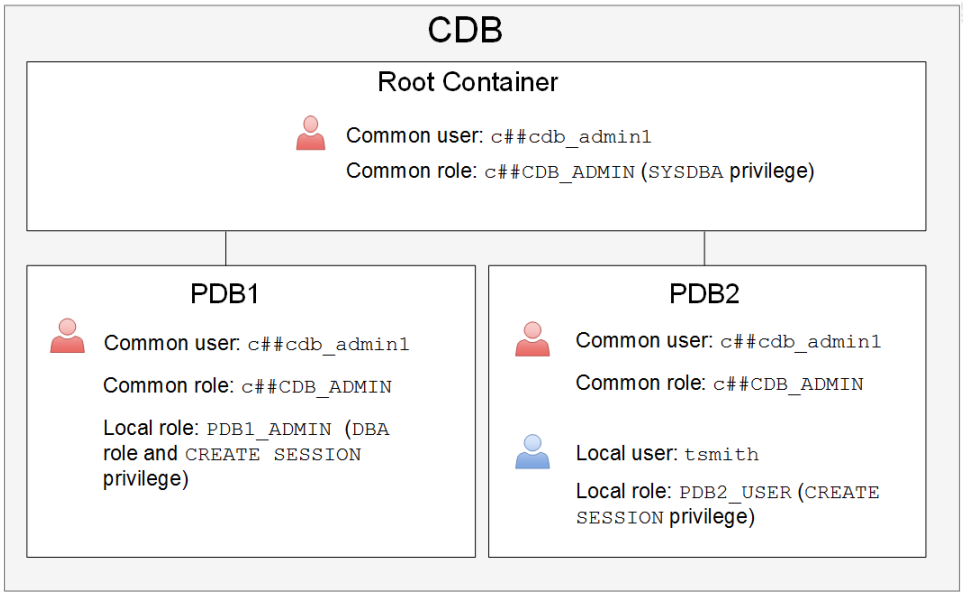
- Locally: Grant the role to a user (or role) in one PDB only.

```
SQL> CONNECT SYS@PDB1 AS SYSDBA
SQL> GRANT <common or local role> TO <common or local user>;
```

There are two ways to grant a role in a multitenant architecture:

- **Commonly:** You grant the role to the user (or role) in all containers. Do this if the user needs to perform the same operation in all containers. To grant a role commonly, log in to the root container or an application root container and issue the `GRANT` command to a common user (or common role) with the `CONTAINER=ALL` clause. The role that you grant must be a common role before you can grant it commonly.
- **Locally:** You grant the role to a user (or role) in one PDB only. To grant a role locally, log in to the PDB where the user exists and issue the `GRANT` command without the `CONTAINER=ALL` clause. The role that you grant can be a common or local role.

Granting Roles: Example



The diagram in the slide illustrates:

- A common role named `c##CDB_ADMIN` is created in all containers. This role consists of the `SYSDBA` privilege and is created for users who need to perform maintenance operations on the entire CDB. The common user, `c##c_admin1`, who exists in every container, is granted the `c##CDB_ADMIN` role commonly, meaning `c##c_admin1` is granted that role in every container.
- A local role named `PDB1_ADMIN` is created and available in `PDB1` only. This role consists of the `DBA` role and the `CREATE SESSION` privilege and is created for users who manage `PDB1`. The common user named `c##c_admin1` is granted this role only in `PDB1`.
- A local role named `PDB2_USER` is created and available in `PDB2` only. The local user named `tsmith`, who exists only in `PDB2`, is granted the `PDB2_USER` role.

Making Roles More Secure

- Roles are usually enabled by default, which means that if a role is granted to a user, then that user can exercise the privileges given to the role immediately.
- Default roles are assigned to the user at connect time.
- Use the following security measures to make roles more secure:
  - Make a role nondefault.
  - Use role authentication.
  - Create application roles.

You can use the following security measures to make roles more secure:

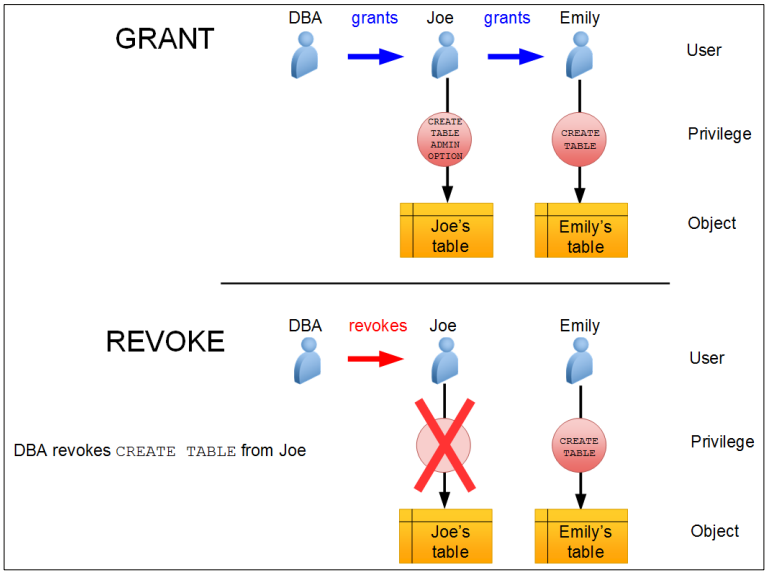
- Make a role nondefault. Not all privileges and roles granted to a user need to be available to that user at logon. You can configure specific roles to be enabled by default and set the other roles to be nondefault (disabled). The user can enable the nondefault roles when needed by using the `SET ROLE` command. For example, user `tsmith` is granted both the `HRCLERK` and `HRMANAGER` roles on a PDB. In addition, you want to automatically enable only the granted `HRCLERK` role after logon. The `HRMANAGER` role has more privileges and contains the `HRCLERK` role. To accomplish this, you configure `tsmith's` default role to be `HRCLERK`, so when `tsmith` logs in, only privileges from the `HRCLERK` role are enabled. When she needs to operate as the manager, she enables the `HRMANAGER` role.
- To configure default roles for a user, use the `ALTER USER` command with the `DEFAULT ROLE` clause. In the `DEFAULT ROLE` clause, you cannot specify a role that is a member of another role (that is, a subrole).
  - Note that the `SET ROLE` command disables roles you don't specify in the command.
- Use role authentication. Have a role require additional authentication by using the `IDENTIFIED` clause to indicate that a user must be authorized by a specified method before the role is enabled with the `SET ROLE` statement. The default authentication for a role is `None`. You can define role authentication in Enterprise Manager Cloud Control, but not in Enterprise Manager Database Express.
- Create application roles. Create secure application roles that a user must enable by executing a PL/SQL procedure successfully. The PL/SQL procedure can check things, such as the user's network address, the program that the user is running, the time of day, and other elements needed, to properly secure a group of permissions.

```
CREATE ROLE secure_application_role IDENTIFIED USING
<security_procedure_name>
```

Revoking Roles and Privileges

- You can use the `REVOKE` statement to:
  - Revoke system privileges from users and roles
  - Revoke roles from users, roles, and program units
  - Revoke object privileges for a particular object from users and roles

Granting and Revoking System Privileges



System privileges that have been granted directly with a `GRANT` command can be revoked by using the `REVOKE` command in SQL*Plus. Users with the `ADMIN OPTION` for a system privilege can revoke the privilege from any other database user. The revoker does not have to be the same user who originally granted the privilege.

There are no cascading effects when a system privilege is revoked, regardless of whether it is given the `ADMIN OPTION`.

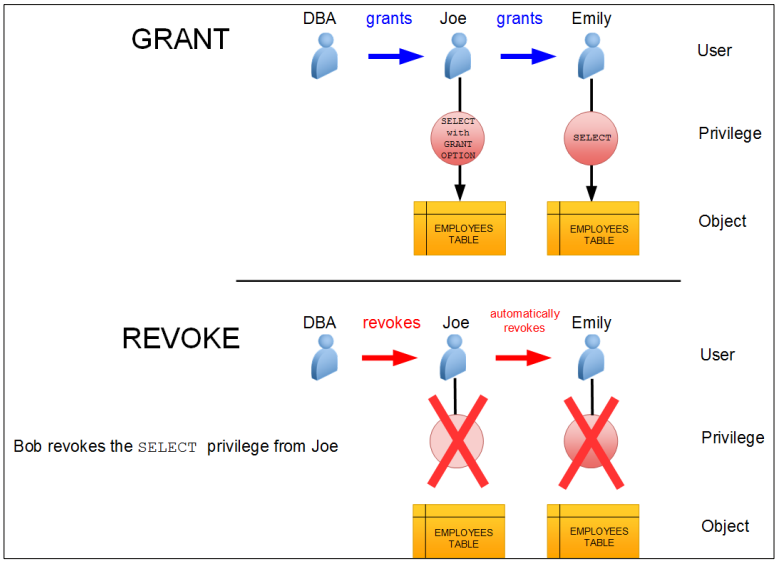
The SQL syntax for revoking system privileges is:

```
SQL> REVOKE <system_privilege> FROM <grantee clause>
```

The diagram in the slide illustrates the following events:

1. The DBA grants the `CREATE TABLE` system privilege to the user Joe with `ADMIN OPTION`.
2. Joe creates a table.
3. Joe grants the `CREATE TABLE` system privilege to the user Emily.
4. Emily creates a table.
5. The DBA revokes the `CREATE TABLE` system privilege from Joe.
6. The result is that Joe's table still exists and he can still access it, but he can't create new tables. Emily's table still exists, and she still has the `CREATE TABLE` system privilege.

Granting and Revoking Object Privileges



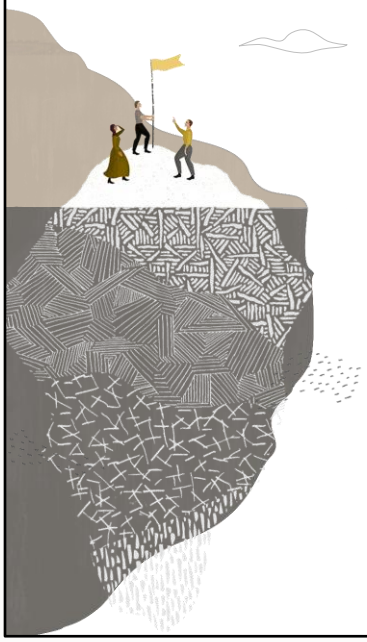
Cascading effects can be observed when revoking a system privilege that is related to a data manipulation language (DML) operation. For example, if the `SELECT ANY TABLE` privilege is granted to a user, and if that user has created procedures that use the table, all procedures that are contained in the user's schema must be recompiled before they can be used again.

Revoking object privileges also cascades when given with `GRANT OPTION`. As a user, you can revoke only those privileges that you have granted. For example, Bob cannot revoke the object privilege that Joe granted Emily. Only the grantee or a user with the privilege called `GRANT ANY OBJECT PRIVILEGE` can revoke object privileges.

The diagram in the slide illustrates object privileges being revoked. Assume that the following events occur:

1. A DBA grants Joe the `SELECT` object privilege on the `EMPLOYEES` table with the `GRANT OPTION`.
2. Joe grants the `SELECT` privilege on the `EMPLOYEES` table to Emily.
3. The DBA revokes the `SELECT` privilege from Joe.
4. The result is that the revoke takes away Joe's ability to access the `EMPLOYEES` table, and the revoke is cascaded to Emily as well.

Summary



- Grant system, schema, and object privileges to database users, commonly and locally
- Create roles
- Grant roles to users and other roles, commonly and locally
- Revoke privileges and roles from users and other roles

ORACLE
University

Configuring User Resource Limits

Objectives

Create and assign profiles to:

- Control resource consumption
- Manage account status and password expiration

Use Oracle-supplied password functions in profiles

Profiles and Users

- Users are assigned only one profile at a time.
- Profiles:
  - Control resource consumption
  - Manage account status and password expiration
- `RESOURCE_LIMIT` must be set to `TRUE` (default) for profiles to impose resource limitations.

Profiles impose a named set of resource limits on database usage and instance resources. Profiles also manage the account status and place limitations on users' passwords (length, expiration time, and so on). Every user is assigned a profile and may belong to only one profile at any given time. If users have already logged in when you change their profile, the change does not take effect until their next login.

The `DEFAULT` profile serves as the basis for all other profiles. Limitations for a profile can be implicitly specified (as in CPU/Session), can be unlimited (as in CPU/Call), or can reference whatever setting is in the `DEFAULT` profile (as in Connect Time).

Profiles cannot impose resource limitations on users unless the `RESOURCE_LIMIT` initialization parameter is set to `TRUE`, which is the default value. Profile password settings are always enforced.

Creating Profiles in a Multitenant Architecture

- Common profile: Replicated in all current and future containers

```
SQL> CREATE PROFILE c##cprofile_dev
  2  limit … CONTAINER=ALL;
```

- Local profile: Created in a single PDB and used within that PDB only

```
SQL> CREATE PROFILE lprofile_PDB1
  2  limit … ;
```

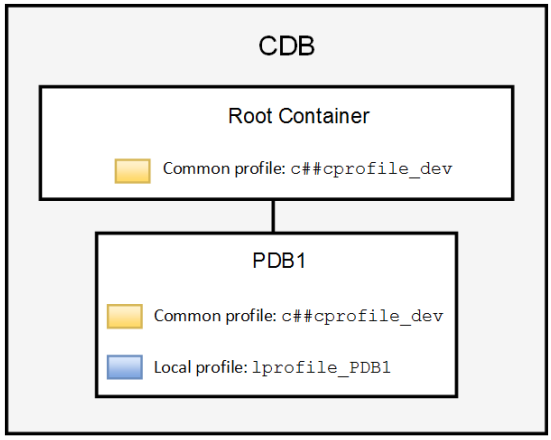
In a multitenant environment, you can create two types of profiles:

- **Common profile:** The profile is replicated in all current and future containers. To create a common profile by using SQL*Plus, log in to the root container or an application root container and issue the `CREATE PROFILE` command with the `CONTAINER=ALL` clause.
- **Local profile:** The profile is created in a single PDB and can be used within that PDB only. To create a local profile by using SQL*Plus, log in to the PDB and issue the `CREATE PROFILE` command without the `CONTAINER=ALL` clause.

In Enterprise Manager Database Express, you cannot drop a profile that is used by users. However, if you drop a profile with the `CASCADE` option (for example, in SQL*Plus), all users who have that profile are automatically assigned the `DEFAULT` profile.

If users have already logged in when you change their profile, the change does not take effect until their next login.

Creating Profiles: Example



In the diagram in the slide, the common profile named `c##cprofile_dev` is created commonly at the CDB level. The `CREATE` operation is replicated in all containers, including the root container where it was initially created. Consequently, the same profile `c##cprofile_dev` is created in `PDB1`. The profile named `lprofile_PDB1` is created locally in `PDB1` and exists only in `PDB1`.

Profile Parameters: Resources

- In a profile, you can control:
  - CPU resources
    - May be limited to a per-session or per-call basis
  - Network and memory resources
    - Connect time, Idle time, Concurrent sessions, Private SGA
  - Disk I/O resources:
    - Limit the amount of data a user can read
    - per-session level or per-call level.

CPU Resources

CPU resources may be limited on a per-session or per-call basis. A CPU/Session limitation of 1,000 means that if any individual session that uses this profile consumes more than 10 seconds of CPU time (CPU time limitations are in hundredths of a second), that session receives the following error message and is logged off:

```
ORA-02392: exceeded session limit on CPU usage, you are being logged off.
```

A per-call limitation does the same thing, but instead of limiting the user's overall session, it prevents any single command from consuming too much CPU. If CPU/Call is limited and the user exceeds the limitation, the command aborts. The user receives an error message, such as:

```
ORA-02393: exceeded call limit on CPU usage.
```

Network and Memory Resources

Each database session consumes system memory resources and (if the session is from a user who is not local to the server) network resources. You can specify the following:

- **Connect Time:** Indicates for how many minutes a user can be connected before being automatically logged off
- **Idle Time:** Indicates for how many minutes a user's session can remain idle before being automatically logged off. Idle time is calculated for the server process only. It does not take into account application activity. The `IDLE_TIME` limit is not affected by long-running queries and other operations.

- **Concurrent Sessions:** Indicates how many concurrent sessions can be created by using a database user account
- **Private SGA:** Limits the amount of space consumed in the System Global Area (SGA) for sorting, merging bitmaps, and so on. This restriction takes effect only if the session uses a shared server configuration.

Disk I/O Resources

Disk I/O resources limit the amount of data a user can read at the per-session level or per-call level. Reads/Session and Reads/Call place a limitation on the total number of reads from both memory and the disk. This can be done to ensure that no I/O-intensive statements overuse memory and disks.

Profile Parameters: Resources

- Profiles cannot
  - Impose resource limitations on users
    - Unless the `RESOURCE_LIMIT` initialization parameter is set to `TRUE`.
  - `RESOURCE_LIMIT` default is `FALSE`
    - Profile resource limitations are ignored.
- Profiles also allow composite limits
  - Based on weighted combinations
    - CPU/session, reads/session, connect time, and private SGA.

CPU Resources

CPU resources may be limited on a per-session or per-call basis. A CPU/Session limitation of 1,000 means that if any individual session that uses this profile consumes more than 10 seconds of CPU time (CPU time limitations are in hundredths of a second), that session receives the following error message and is logged off:

```
ORA-02392: exceeded session limit on CPU usage, you are being logged off.
```

A per-call limitation does the same thing, but instead of limiting the user's overall session, it prevents any single command from consuming too much CPU. If CPU/Call is limited and the user exceeds the limitation, the command aborts. The user receives an error message, such as:

```
ORA-02393: exceeded call limit on CPU usage.
```

Network and Memory Resources

Each database session consumes system memory resources and (if the session is from a user who is not local to the server) network resources. You can specify the following:

- **Connect Time:** Indicates for how many minutes a user can be connected before being automatically logged off
- **Idle Time:** Indicates for how many minutes a user's session can remain idle before being automatically logged off. Idle time is calculated for the server process only. It does not take into account application activity. The `IDLE_TIME` limit is not affected by long-running queries and other operations.

Profile Parameters: Locking and Passwords

- In a profile, specific parameters control
  - Account locking
  - Password aging
  - Expiration
  - Password history.
- Profile password settings are always enforced.
- Account locking enables automatic locking of accounts for a set duration when
  - Specified number of failed logon attempts
  - Account inactivity for a predefined number of days

Account Locking

Account locking enables automatic locking of accounts for a set duration when users fail to log in to the system in the specified number of attempts or when accounts sit inactive for a predefined number of days (users have not attempted to log in to their accounts). Configure the following profile parameters:

- `FAILED_LOGIN_ATTEMPTS` specifies the number of failed login attempts before the lockout of the account.
- `PASSWORD_LOCK_TIME` specifies the number of days for which the account is locked after the specified number of failed login attempts.
- `INACTIVE_ACCOUNT_TIME` specifies the number of days an account can be inactive before it is locked.

Password History

Password history checks the new password to ensure that the password is not reused for a specified amount of time or a specified number of password changes. Configure one of the following parameters:

- `PASSWORD_REUSE_TIME` specifies that a user cannot reuse a password for a given number of days.
- `PASSWORD_REUSE_MAX` specifies the number of password changes that are required before the current password can be reused.
- `PASSWORD_VERIFY_FUNCTION` checks for password complexity for the `SYS` user.

Recall that the values of the profile parameters are either set or inherited from the `DEFAULT` profile.

If both password history parameters have a value of `UNLIMITED`, Oracle Database ignores both. The user can reuse any password at any time, which is not a good security practice. If both parameters are set, password reuse is allowed, but only after meeting both conditions. The user must have changed the password the specified number of times, and the specified number of days must have passed since the old password was last used. For example, the profile of user `ALFRED` has `PASSWORD_REUSE_MAX` set to `10` and `PASSWORD_REUSE_TIME` set to `30`. This means user `ALFRED` cannot reuse a password until he has reset the password 10 times and until 30 days have passed since the password was last used. If one parameter is set to a number and the other parameter is specified as `UNLIMITED`, then the user can never reuse a password.

Profile Parameters: Locking and Passwords

- Password aging and expiration enables user passwords
  - A lifetime for remaining valid after password expiration
    - Must be changed after Lifetime has expired
- Password history checks ensures
  - New password is not reused
    - For a specified amount of time
    - Or for a specified number of password changes
- Password complexity verification
  - Verifies passwords meet defined rules

Password Aging and Expiration

Password aging and expiration enables user passwords to have a lifetime, after which the passwords expire and must be changed. Configure the following profile parameters:

- `PASSWORD_LIFE_TIME` determines the lifetime of the password in days, after which the password expires.
- `PASSWORD_GRACE_TIME` specifies a grace period in days for changing the password after the first successful login after the password has expired.

Expiring passwords and locking the `SYS`, `SYSMAN`, and `DBSNMP` accounts prevent Enterprise Manager from functioning properly. The applications must catch the "password expired" warning message and handle the password change; otherwise, the grace period expires and the user is locked out without knowing the reason.

Password Complexity Verification

Password complexity verification makes a complexity check on the password to verify that it meets certain rules. The check must ensure that the password is complex enough to provide protection against intruders who may try to break into the system by guessing the password.

The `PASSWORD_VERIFY_FUNCTION` parameter names a PL/SQL function that performs a password complexity check before a password is assigned. Password verification functions must be owned by the `SYS` user and return a Boolean value (`TRUE` or `FALSE`). A model password verification function is provided in the `utlpwdmg.sql` script found in the following directories:

- UNIX and Linux platforms: `$ORACLE_HOME/rdbms/admin`
- Windows platforms: `%ORACLE_HOME%\rdbms\admin`

Oracle-Supplied Password Verification Functions

- Complexity verification checks that each password is complex enough to provide reasonable protection against intruders who try to break into the system by guessing passwords.
- You can create your own password verification functions.
- Oracle Database provides the following functions by default:
  - ORA12*c*_VERIFY_FUNCTION
  - ORA12*c*_STRONG_VERIFY_FUNCTION
  - ORA12*c*_STIG_VERIFY_FUNCTION
- Password complexity checking is not enforced for the SYS user.

Complexity Verification

Complexity verification checks that each password is complex enough to provide reasonable protection against intruders who try to break into the system by guessing passwords. Using a complexity verification function forces users to create strong, secure passwords for database user accounts. You must ensure that the passwords for your users are complex enough to provide reasonable protection against intruders who try to break into the system by guessing passwords.

Oracle-Supplied Functions

You can create your own password verification functions; however, Oracle Database provides the following functions by default:

- ORA12*c*_VERIFY_FUNCTION: This function performs the minimum complexity checks such as checking for a minimum password length and that the password is not the same as the username.
- ORA12*c*_STRONG_VERIFY_FUNCTION: This function provides a stronger password complexity function that takes into consideration recommendations from the US Department of Defense Database Security Technical Implementation Guide.
- ORA12*c*_STIG_VERIFY_FUNCTION: This function fulfills the requirements of the Security Technical Implementation Guides (STIG) and is the default handler for the ORA_STIG_PROFILE profile.

Assigning Profiles in a Multitenant Architecture

- Commonly: The profile assignment is replicated in all current and future containers.

```
SQL> CONNECT / AS SYSDBA
SQL> ALTER USER <common user> PROFILE <common profile> CONTAINER=ALL;
```

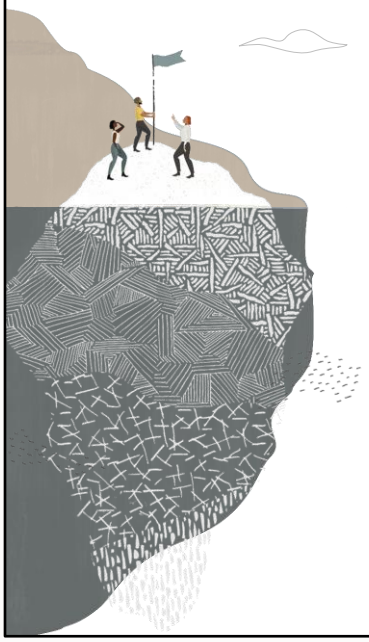
- Locally: The profile assignment occurs in one PDB (stand-alone or application container) only.

```
SQL> CONNECT SYS@PDB1 AS SYSDBA
SQL> ALTER USER <common or local user> PROFILE <common or local profile>;
```

There are two ways to assign a profile to a user:

- **Commonly:** The profile assignment is replicated in all current and future containers. The user must be a common user and the profile must be a common profile. Do this if the user needs the same profile in all containers. To assign a profile commonly, log in to the root container and issue the `ALTER USER` command with the `PROFILE` and `CONTAINER=ALL` clauses.
- **Locally:** The profile assignment occurs in one PDB (stand-alone or application container) only. The user can be common or local, and the profile can be common or local. To assign a profile locally, log in to the PDB where the user exists and issue the `ALTER USER` command with the `PROFILE` clause. Exclude the `CONTAINER=ALL` clause.

Summary



Create and assign profiles to:

- Control resource consumption
- Manage account status and password expiration

Use Oracle-supplied password functions in profiles

ORACLE
University

Implementing Oracle Database Auditing

Objectives

- Describe DBA responsibilities for security and auditing
- Enable unified auditing
- Create unified audit policies
- Maintain the audit trail

Database Security

- A secure system ensures the confidentiality of the data that it contains. There are several aspects of security:
  - Restricting access to data and services
  - Authenticating users
  - Monitoring for suspicious activity

Oracle Database provides the industry's best framework for a secure system. But for that framework to be effective, the database administrator must follow best practices and continually monitor database activity.

Restricting Access to Data and Services

All users must not have access to all data. Depending on what is stored in your database, restricted access can be mandated by business requirements, customer expectations, and (increasingly) legal restrictions. Credit card information, health-care data, identity information, and so on must be protected from unauthorized access. The Oracle Database server provides extremely fine-grained authorization controls to limit database access. Restricting access must include applying the principle of least privilege.

Authenticating Users

To enforce access controls on sensitive data, the system must first know who is trying to access the data. Compromised authentication can render all other security precautions useless. The most basic form of user authentication is challenging users to provide something that they know, such as a password. Ensuring that passwords follow simple rules can greatly increase the security of your system. Stronger authentication methods include requiring users to provide something that they have, such as a token or public key infrastructure (PKI) certificate. An even stronger form of authentication is to identify users through a unique biometric characteristic such as a fingerprint, iris scan, bone structure patterns, and so on. The Oracle Database server supports advanced authentication techniques (such as token-, biometric-, and certificate-based identification) through the Oracle Advanced Security option. User accounts that are not in use must be locked to prevent attempts to compromise authentication.

Monitoring for Suspicious Activity

Even authorized and authenticated users can sometimes compromise your system. Identifying unusual database activity (such as an employee who suddenly begins querying large amounts of credit card information, research results, or other sensitive information) can be the first step in detecting information theft. The Oracle Database server provides a rich set of auditing tools to track user activity and identify suspicious trends.

Monitoring for Compliance

- Monitoring or auditing must be an integral part of your security procedures.
- Review the following:
  - Mandatory auditing
  - Standard database auditing
  - Value-based auditing
  - Fine-grained auditing (FGA)

Auditing, which means capturing and storing information about what is happening in the system, increases the amount of work the system must do. Auditing must be focused so that only events that are of interest are captured. Properly focused auditing has minimal impact on system performance. Improperly focused auditing can significantly affect performance.

- **Mandatory auditing:** All Oracle databases audit certain actions regardless of other audit options or parameters. The reason for mandatory audit logs is that the database needs to record some database activities, such as connections by privileged users.
- **Standard database auditing:** Select the objects and privileges that you want to audit and create the appropriate audit policies.
- **Value-based auditing:** Extend standard database auditing, capturing not only the audited event that occurred but also the actual values that were inserted, updated, or deleted. Value-based auditing is implemented through database triggers.
- **Fine-grained auditing (FGA):** Extend standard database auditing, capturing the actual SQL statement that was issued rather than only the fact that the event occurred.

Types of Activities to be Audited

- You can audit the following types of activities:
  - User accounts, roles, and privileges
  - Object actions
  - Application context values
  - Oracle Data Pump
  - Oracle Database Real Application Security
  - Oracle Database Vault
  - Oracle Label Security
  - Oracle Recovery Manager
  - Oracle SQL*Loader direct path events

Through the use of auditing policies, you can configure audit settings for the following activities:

- Logging on to the database and using privileges and roles
- Executing SQL statements against specific database objects
- Application context values
- Utilities and features:
  - Oracle Data Pump
  - Oracle Database Real Application Security
  - Oracle Database Vault
  - Oracle Label Security
  - Oracle Recovery Manager
  - Oracle SQL*Loader Direct Load

Mandatorily Audited Activities

- The following activities are audited:
  - CREATE/ALTER/DROP AUDIT POLICY
  - AUDIT/NOAUDIT
  - EXECUTE of:
    - DBMS_FGA
    - DBMS_AUDIT_MGMT
  - ALTER TABLE against AUDSYS audit trail table
  - Top-level statements by administrative users (SYS, SYSDBA, SYSOPER, SYSASM, SYSBACKUP, SYSDG, and SYSKM) until the database opens

The following audit-related activities are mandatorily audited:

- CREATE/ALTER/DROP AUDIT POLICY
- AUDIT/NOAUDIT
- EXECUTE of the DBMS_FGA PL/SQL package
- EXECUTE of the DBMS_AUDIT_MGMT PL/SQL package
- ALTER TABLE attempts on the AUDSYS audit trail table
- Top-level statements by SYS, SYSDBA, SYSOPER, SYSASM, SYSBACKUP, SYSDG, and SYSKM until the database opens. After the database opens, these users are audited based on the defined audit settings.

Understanding Auditing Implementation

- **Mixed mode auditing** is the default when a new database is created.
- Mixed mode auditing enables the use of:
  - Pre-Oracle Database 12*c* auditing features
  - **Unified auditing** features
- The recommendation from Oracle is to migrate to pure unified auditing.
- Query `V$OPTION` to determine if the database has been migrated to unified auditing:

```
SELECT VALUE FROM V$OPTION WHERE PARAMETER = 'Unified Auditing';

PARAMETER         VALUE
----------------- ----------
Unified Auditing TRUE
```

Prior to Oracle Database 12*c*, audit records from various sources were stored in different locations. **Unified auditing,** in which all audit records are stored in a single audit table, was introduced in Oracle Database 12c.

When you create a new database, mixed mode auditing is enabled. This mode enables you to use the auditing features available before Oracle Database 12c and also the unified auditing features. Mixed mode auditing is enabled by default through the `ORA_SECURECONFIG` predefined auditing policy for newly created databases.

Note: Oracle recommends that you migrate to pure unified auditing. See the *Oracle Database Upgrade Guide* for details.

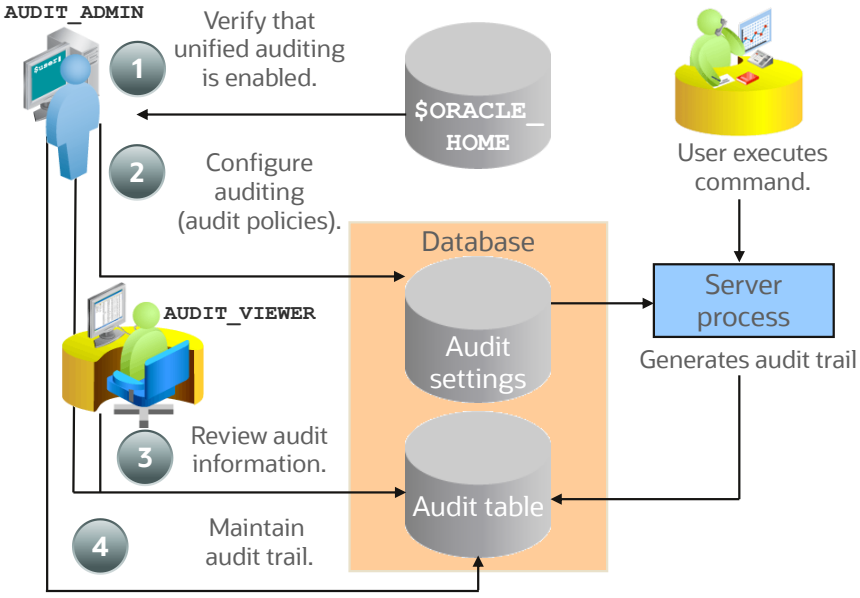
Administering the Roles Required for Auditing

- A user must be granted one of the following roles to perform auditing:
  - AUDIT_ADMIN enables the user to:
    - Create unified and fine-grained audit policies
    - Execute the AUDIT and NOAUDIT SQL statements
    - View audit data
    - Manage the audit trail (table in the AUDSYS schema)
  - AUDIT_VIEWER enables the user to:
    - View and analyze audit data
    - Execute the DBMS_AUDIT_UTIL PL/SQL package

Users must be granted the appropriate privilege to configure auditing and view audit data. To support separation of duty, two default roles are provided:

- AUDIT_ADMIN: Enables the grantee to configure auditing settings, create and administer audit policies (unified and find-grained), and view and analyze audit data. This role is typically granted to a security administrator.
- AUDIT_VIEWER: Enables the grantee to view and analyze audit data. This role is typically granted to external auditors.

Database Auditing: Overview



Auditing is enabled by default. You can query `V$OPTION` to verify that unified auditing is enabled as shown in this example:

```
SELECT VALUE FROM V$OPTION WHERE PARAMETER = 'Unified Auditing';

PARAMETER         VALUE

----------------- ----------

Unified Auditing TRUE
```

Note: By default, unified auditing is not enabled for *upgraded* databases. If you have upgraded from a release prior to Oracle Database 12*c*, your database uses the same auditing functionality that was used in the earlier release. After you complete the migration to unified auditing, traditional auditing is disabled, and the new audit records write to the unified audit trail. See the *Oracle Database Upgrade Guide* for detailed information on migrating to unified auditing.

You must create and enable unified audit policies to specify which activities should be audited.

When a user executes a command or performs an activity that is defined in an auditing policy, audit records are generated. The audit records are written to an internal relational table in the `AUDSYS` schema and can be viewed by querying the `UNIFIED_AUDIT_TRAIL` view.

Maintaining the audit trail is an important administrative task. Depending on the focus of the audit options, the audit trail can grow very large very quickly. If not properly maintained, the audit trail can create so many records that it affects the performance of the system. Audit overhead is directly related to the number of records that are produced.

Detailed information on all these steps is provided later in this lesson.

Configuring Auditing

Method	Description
Unified audit policies	Group audit settings into a policy
Predefined unified audit policies	Commonly used security-relevant audit settings
Fine-grained audit policies	Define specific conditions that must be met for auditing to take place

You can configure auditing by grouping audit settings into a unified audit policy.

Oracle Database includes predefined unified audit policies, which include commonly used security audit settings. See *Oracle Database Security Guide* for a list of predefined audit policies.

Fine-grained audit policies enable you to define specific conditions that generate an audit record.

Additional information about these methods is provided later in the lesson.

Creating a Unified Audit Policy

- Use the `CREATE AUDIT POLICY` statement to create a unified audit policy:

```
CREATE AUDIT POLICY select_emp_pol
ACTIONS select on hr.employees
```

- To simplify policy management, group related options into a single policy.

Use the `CREATE AUDIT POLICY` statement to create a unified audit policy. The unified audit policy syntax is designed so that you can create one policy that covers all the audit settings that your database needs. A good practice is to group related options into a single policy instead of creating multiple small policies. This enables you to manage the policies more easily.

The example in the slide shows how to create an audit policy named `SELECT_EMP_POL`. This policy specifies that `SELECT` statements against the `HR.EMPLOYEES` table will be audited.

You can also create audit policies by using Enterprise Manager Cloud Control.

Creating an Audit Policy: Systemwide Audit Options

- System privileges:

```
CREATE AUDIT POLICY audit_syspriv_pol1
PRIVILEGES SELECT ANY TABLE, CREATE LIBRARY
```

- Actions:

```
CREATE AUDIT POLICY audit_actions_pol2
ACTIONS AUDIT, ALTER TRIGGER
```

- Roles:

```
CREATE AUDIT POLICY audit_role_pol3
ROLES mgr_role
```

- System privileges, actions, and roles:

```
CREATE AUDIT POLICY audit_mixed_pol4
PRIVILEGES DROP ANY TABLE
ACTIONS    CREATE TABLE, DROP TABLE, TRUNCATE TABLE
ROLES      emp_role
```

You can create an audit policy with systemwide or object-specific audit options.

The systemwide options can be of three types:

- The privilege audit option audits all events that exercise the specified system privilege, as in the first example in the slide.
- The action audit option indicates which RDBMS action should be audited, such as the `ALTER TRIGGER` action in the second example.
- The role audit option audits the use of all system or object privileges granted directly to the `MGR_ROLE` role, as in the third example.

You can configure privilege, action, and role audit options together in the same audit policy, as shown in the fourth example.

You can find a list of auditable systemwide options in the `SYS.AUDITABLE_SYSTEM_ACTIONS` table.

Creating an Audit Policy: Object-Specific Actions

- Create audit policies based on object-specific options.

```
CREATE AUDIT POLICY audit_objpriv_pol5
ACTIONS SELECT, UPDATE, LOCK ON hr.employees
```

```
CREATE AUDIT POLICY audit_objpriv_pol6
ACTIONS ALL
```

```
CREATE AUDIT POLICY audit_objpriv_pol7
ACTIONS EXECUTE, GRANT ON hr.raise_salary_proc
```

Object-specific options are the second type of audit options. These options are actions that are specific to objects in the database.

The first example in the slide creates an audit policy to audit any select and update action on any object, and any lock on the `HR.EMPLOYEES` table.

The second example creates an audit policy to audit any object-specific action on any object.

The third example creates an audit policy to audit any execute action on any procedural object, and any grant on the `HR.RAISE_SALARY_PROC` procedure.

Object-level audit options are dynamic. That is, changes in these options become applicable for current and subsequent user sessions.

Creating an Audit Policy: Specifying Conditions

- Condition and evaluation **PER SESSION**

```
CREATE AUDIT POLICY audit_mixed_pol5
ACTIONS RENAME ON hr.employees,ALTER  ON hr.jobs,
WHEN 'SYS_CONTEXT (''USERENV'', ''SESSION_USER'')=''JIM'''
EVALUATE PER SESSION
```

- Condition and evaluation **PER STATEMENT**

```
CREATE AUDIT POLICY audit_objpriv_pol6
ACTIONS ALTER ON OE.ORDERS
WHEN 'SYS_CONTEXT(''USERENV'',''CLIENT_IDENTIFIER'')=''OE'''
EVALUATE PER STATEMENT
```

- Condition and evaluation **PER INSTANCE**

```
CREATE AUDIT POLICY audit_objpriv_pol7
ROLES dba
WHEN SYS_CONTEXT(''USERENV'',''INSTANCE_NAME'')=''sales'''
EVALUATE PER INSTANCE
```

Audit policies can evaluate the condition per statement, once per session or once per instance.

The first example in the slide creates an audit policy to audit any rename action on the `HR.EMPLOYEES` table, and any alter action on the `HR.JOBS` table, provided that the user executing the audited statement is `JIM`. The condition is evaluated only once in the session.

The second example creates an audit policy to audit any alter action on the `OE.ORDERS` table, provided that the user executing the audited statement is the schema owner `OE`. The condition is evaluated each time an `ALTER` statement is executed on the `OE.ORDERS` table.

The third example creates an audit policy to audit any privilege granted to the DBA role, provided that the instance name is "`sales`." The condition is evaluated only once during the database instance lifetime. After Oracle Database evaluates the condition, it caches and reuses the result for the remainder of the instance lifetime.

Enabling and Disabling Audit Policies

- Enable audit policies:
  - Apply to all users.

```
AUDIT POLICY audit_syspriv_pol1;
```

  - Apply only to some users.

```
AUDIT POLICY audit_pol2 BY scott, oe;
AUDIT POLICY audit_pol3 BY sys;
```

  - Exclude some users.

```
AUDIT POLICY audit_pol4 EXCEPT jim, george;
```

  - Audit the recording based on failed or succeeded actions.

```
AUDIT POLICY audit_syspriv_pol1 WHENEVER SUCCESSFUL ;
AUDIT POLICY audit_objpriv_pol2 WHENEVER NOT SUCCESSFUL ;
```

```
AUDIT POLICY auditpol5 BY joe WHENEVER SUCCESSFUL ;
```

- Disable audit policies by using the `NOAUDIT` command.

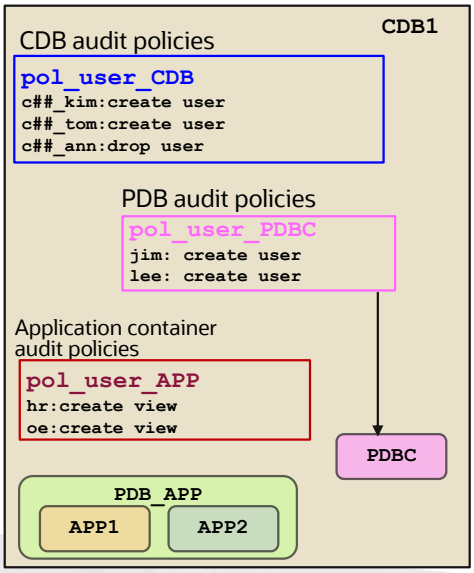
After creating the audit policy, you must enable it by using the `AUDIT` statement.

All users are audited by default, except if you define a list of those audited. Apply the audit policy to one or more users by using the `BY` clause. The audit administrator audits `SYS`-user-specific actions the same way as other user actions. Exclude some users by using the `EXCEPT` clause.
You cannot have a `BY` list and an `EXCEPT` list in the same policy enablement statement

Audit records are generated whether the user's actions failed or succeeded. If you want to audit actions only when the user's actions failed, use the `WHENEVER NOT SUCCESSFUL` clause. If you want to audit actions only when the user's actions succeeded, use the `WHENEVER SUCCESSFUL` clause. When you omit the `WHENEVER` clause, the statement is audited whether the action is successful or not, and the `RETURN_CODE` column displays whether the action succeeded or not.

To disable an audit policy, use the `NOAUDIT` command.

Auditing Actions in the CDB and PDBs



1. Connect to the CDB root, to an application root, or to a regular PDB.
2. Create common or local unified audit policies:
   - For all PDBs (*connect to CDB root*)
   - For all application PDBs of an application container (*connect to the application root*)
   - For a regular PDB or a specific application PDB (*connect to the PDB*)
3. Enable/disable audit policies:
   - Define users or users being granted roles to be audited (*DBA role*).
   - Use the `AUDIT POLICY` and `NOAUDIT POLICY` commands.

You can create a unified audit policy commonly at the CDB level for the whole CDB. If you create the unified audit policy in the application root, it applies to all application PDBs that belong to the application container. If you create the unified audit policy locally, it applies to the local container, whether it is an application PDB or regular PDB.

The unified audit policy provides the ability to audit the system privileges used or actions performed in all PDBs in the CDB. It also provides the ability to audit object privileges used on common objects by application common users in all application PDBs. Auditing can be in an application container or exclusively in a specific PDB.

Application common unified audit policies need NOT be explicitly synchronized in application PDBs with the application root. An implicit application `BEGIN-END` block is added for application common unified audit policies when the end user does not create them inside an explicit application `BEGIN-END` block. In this case, the creation of application common unified audit policies does not require explicit application `BEGIN-END` block statements. However, application common unified audit policies, when created within an explicit application `BEGIN` statement, will require an explicit application `END` statement.

In the example in the slide, the `POL_USER_CDB` unified audit policy that is created in the CDB root audits any `CREATE` or `DROP USER` statement that is performed by specific users (`c##_kim`, `c##_tom`, and `c##_ann`).

The `POL_USER_APP` unified audit policy that is created in the `PDB_APP` application root audits any `CREATE VIEW` operation performed by application common users (`hr` and `oe`).

The `pol_user_PDBC` unified audit policy that is created in the regular `PDBC` PDB audits any `CREATE` or `DROP USER` performed by local users (`jim`, `lee`, and `bob`).

The audit records are generated in the container's own audit trail, in the container where the action was executed.

A local unified audit policy can be enabled on local and common roles and becomes effective for users to whom the local or common role is granted directly. A common unified audit policy, alternatively, can be enabled only on common roles and becomes effective only for common users to whom the common role is granted. The clause used is `BY USERS WITH GRANTED ROLES role_list`.

A good example of using this capability is the predefined role called `DBA`, which contains most of the system privileges, granted to special privileged users, which might be considered for auditing.

The audit-administrator does not have to enable the unified audit policies on all individual users explicitly. Over a period of time, there could be new users with the `DBA` role granted. Some of the earlier DBA users might no longer have the `DBA` role. The audit-administrator does not have to keep track of such changing auditing requirements and enable the unified audit policies appropriately for a new set of DBA users. Similarly, users who no longer have the `DBA` role granted will automatically be excluded from auditing and, thus, avoid generating unnecessary audit records.

Modifying a Unified Audit Policy

- Enabled and disabled unified audit policies can be modified.
- Use the `ALTER AUDIT POLICY` statement to modify a unified audit policy:

```
ALTER AUDIT POLICY select_emp_pol
ADD ACTIONS select on hr.job_history
```

You can change most properties in a unified audit policy, except a `CONTAINER` setting. You can make changes to enabled and disabled audit policies. You do not need to re-enable an audit policy after altering it.

For an object unified audit policy, the new audit settings take place immediately after it has been altered, for both the active and subsequent user sessions. If you alter system audit options, or audit conditions of the policy, then they are activated for new user sessions but not the current user session.

Auditing Top-Level Statements Only

- Top-level statement unified auditing enables you to:
  - Audit a top-level user or direct user activities in the database without collecting indirect user activity

```
SQL> CREATE AUDIT POLICY actions_all_pol ACTION ALL ONLY TOPLEVEL;
SQL> AUDIT POLICY actions_all_pol BY SYS;
```

```
SQL> CREATE AUDIT POLICY update_emp_pol ACTIONS UPDATE ON HR.EMPLOYEES ONLY TOPLEVEL;
SQL> AUDIT POLICY update_emp_pol;
```

  - Minimize audit records

```
SQL> CONNECT user1@PDB1
SQL> UPDATE hr.employees SET salary = salary * 0.1 WHERE empno = 100;
```

Direct ➔ Audited

```
SQL> EXEC hr.salary_emp_raise (empno => 100, increase => '0.1')
```

Not direct ➔ Not audited

You can audit only top-level user directly issued events, without the overhead of indirect SQL statements. Top-level statements are SQL statements that users directly issue. These statements can be important for both security and compliance. SQL statements run from within PL/SQL procedures or functions are not considered top level because they may be less relevant for auditing purposes.

The `CREATE AUDIT POLICY` statement can include or exclude top-level statement audit records in the unified audit trail for any user by using the `ONLY TOPLEVEL` clause.

The first example in the slide shows how to define an audit policy that captures all top-level statements executed by the `SYS` user.

The second example shows how to limit the audit trail to top-level instances of the `UPDATE` statement on the `HR.EMPLOYEES` table. When a user executes an `UPDATE` command on the `HR.EMPLOYEES` table, the action is audited, whereas when the user executes the same command through a procedure, the action is not audited.

Viewing Audit Policy Information

```
SQL> SELECT policy_name, audit_option, condition_eval_opt
  2  FROM   audit_unified_policies;

POLICY_NAME          AUDIT_OPTION     CONDITION_EVAL_OPT
-------------------- ---------------- ----------------
POL1                 DELETE           INSTANCE
POL2                 TRUNCATE TABLE   NONE
POL3                 RENAME           SESSION
POL4                 ALL ACTIONS      STATEMENT
```

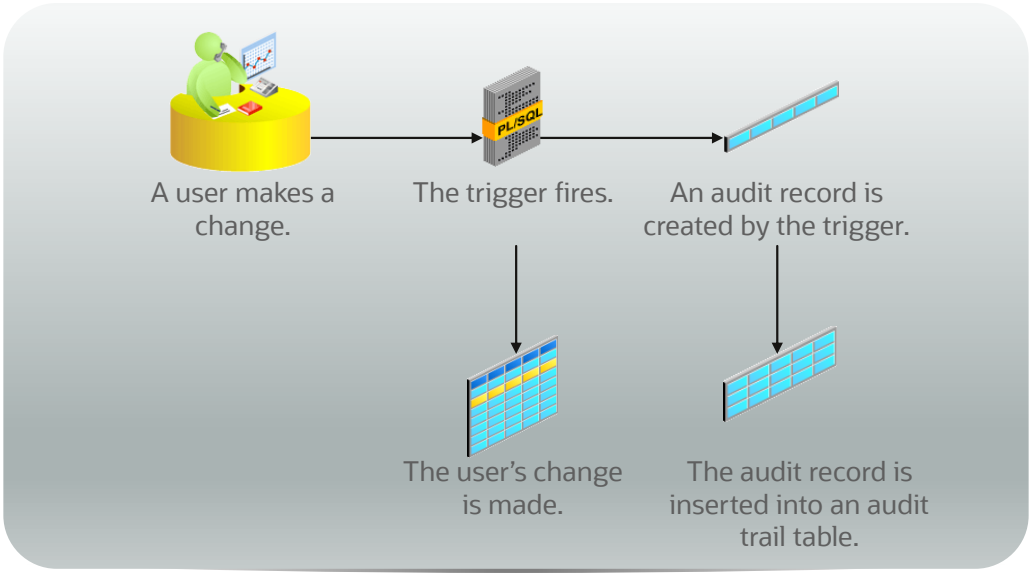
```
SQL> SELECT policy_name, enabled_opt, user_name, success, failure
  2  FROM   audit_unified_enabled_policies;

POLICY_NAME          ENABLED_ USER_NAME  SUC FAI
-------------------- -------- ---------- --- ---
POL3                 BY       PM         NO  YES
POL2                 EXCEPT   SYSTEM     NO  YES
POL4                 BY       SYS        YES
POL6                 BY       ALL USERS  YES NO
```

Query AUDIT_UNIFIED_POLICIES to view a list of existing audit policies. The CONDITION_EVAL_OPT column defines when the condition, if any, is evaluated.

Query AUDIT_UNIFIED_ENABLED_POLICIES to view a list of enabled audit policies. The ENABLED_OPT column indicates whether a list of audited users is defined with the BY value or an exception list of excluded users is defined with the EXCEPT value. The audited or excluded users are listed in the USER_NAME column. The SUCCESS and FAILURE columns indicate whether the policy generates audit records only when the user's actions succeed or fail.

Value-Based Auditing



Value-based auditing leverages database triggers (event-driven PL/SQL constructs) to capture changed values in objects.

When a user inserts, updates, or deletes data from a table with the appropriate trigger attached, the trigger works in the background to copy audit information to a table that is designed to contain the audit information. Value-based auditing tends to degrade performance more than standard database auditing because the audit trigger code must be executed each time the insert, update, or delete operation occurs. The degree of degradation depends on the efficiency of the trigger code. Value-based auditing must be used only in situations in which the information captured by standard database auditing is insufficient.

Value-based auditing is implemented by user or third-party code. The Oracle database provides the PL/SQL constructs to allow value-based audit systems to be built.

The key to value-based auditing is the audit trigger, which is simply a PL/SQL trigger that is constructed to capture audit information.

Example of a typical audit trigger:

```
CREATE OR REPLACE TRIGGER system.hrsalary_audit
       AFTER UPDATE OF salary
       ON hr.employees
       REFERENCING NEW AS NEW OLD AS OLD
       FOR EACH ROW
   BEGIN
    IF :old.salary != :new.salary THEN
        INSERT INTO system.audit_employees
        VALUES (sys_context('userenv','os_user'), sysdate,
        sys_context('userenv','ip_address'),
        :new.employee_id ||
               ' salary changed from '||:old.salary||
        ' to '||:new.salary);
  END IF;
   END;
/
```

This trigger focuses auditing on capturing changes to the salary column of the `HR.EMPLOYEES` table. When a row is updated, the trigger checks the salary column. If the old salary is not equal to the new salary, the trigger inserts an audit record into the `AUDIT_EMPLOYEES` table (created via a separate operation in the `SYSTEM` schema). The audit record includes the username, the IP address from which the change is made, the primary key identifying which record is changed, and the actual salary values that are changed.

Database triggers can also be used to capture information about user connections in cases where standard database auditing does not gather sufficient data. With login triggers, the administrator can capture data that identifies the user who is connecting to the database. Examples include the following:

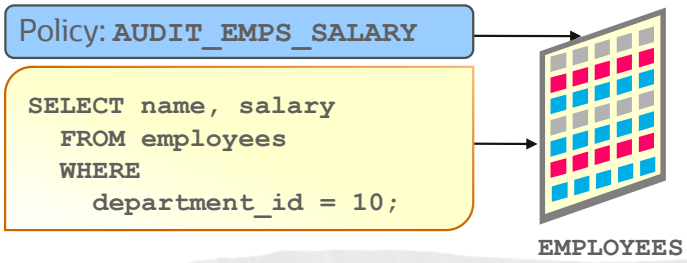
- IP address of the person logging in
- First 48 characters of the program name that is used to connect to the instance
- Terminal name that is used to connect to the instance

For a complete list of user parameters, see the section titled "`SYS_CONTEXT`" in the *Oracle Database SQL Reference*.

Value-based triggers have been superseded in many cases by the fine-grained auditing (FGA) feature.

Fine-Grained Auditing

- Monitors data access on the basis of content
- Audits SELECT, INSERT, UPDATE, DELETE, and MERGE
- Can be linked to one or more columns in a table or view
- May execute a procedure
- Is administered with the DBMS_FGA package



Database auditing records the fact that an operation has occurred but does not capture information about the statement that caused the operation. Fine-grained auditing (FGA) extends that capability to enable the capture of actual SQL statements that query or manipulate data.

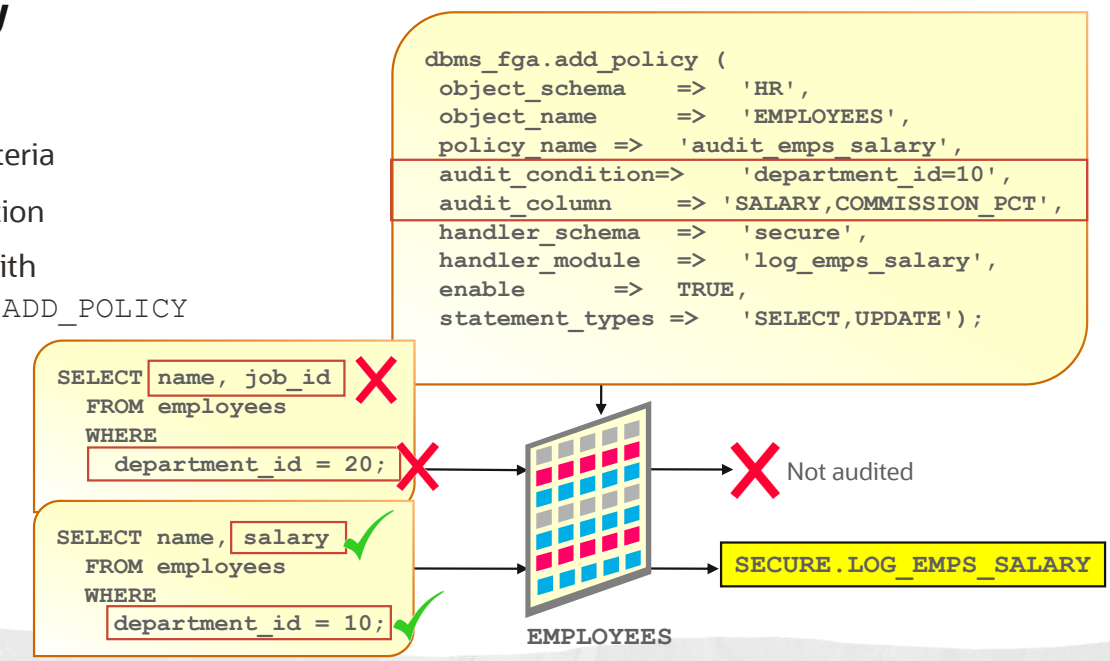
FGA also allows auditing to be more narrowly focused than standard or value-based database auditing.

FGA options can be focused by individual columns in a table or view, and can even be conditional so that audits are captured only if certain administrator-defined specifications are met. More than one relevant column is supported for an FGA policy. By default, if any one of these columns is present in the SQL statement, it is audited. DBMS_FGA.ALL_COLUMNS and DBMS_FGA.ANY_COLUMNS are provided to audit on the basis of whether any or all of the relevant columns are used in the statement.

Use the DBMS_FGA PL/SQL package to create an audit policy on the target table or view. If any of the rows returned from a query block match the audited column and the specified audit condition, an audit event causes an audit record to be created and stored in the audit trail. As an option, the audit event can also execute a procedure. FGA automatically focuses auditing at the statement level. A SELECT statement that returns thousands of rows thus generates only one audit record.

FGA Policy

- Defines:
  - Audit criteria
  - Audit action
- Is created with `DBMS_FGA.ADD_POLICY`



The example in the slide shows the creation of a fine-grained auditing policy with the `DBMS_FGA.ADD_POLICY` procedure, which accepts the following arguments:

- **Policy Name:** You assign each FGA policy a name when you create it. The example in the slide names the policy `AUDIT_EMPS_SALARY` by using the following argument:

  ```
  policy_name => 'audit_emps_salary'
  ```

- **Audit Condition:** The audit condition is a SQL predicate that defines when the audit event must fire. In the example in the slide, all rows in department 10 are audited by using the following condition argument:

  ```
  audit_condition => 'department_id = 10'
  ```

  Note: Fine-grained auditing looks at the result set of the query, so with the FGA policy shown in the slide, queries that return rows matching the policy specifications will cause an audit record to be created. For example, in the query "`select * from employees`", all rows including those having "10" in `DEPARTMENT_ID` may be returned, so an audit row is created.

- **Audit Column:** The audit column defines the data that is being audited. An audit event occurs if this column is included in the `SELECT` statement or if the audit condition allows the selection. The example in the slide audits two columns by using the following argument:

  ```
  audit_column => 'SALARY,COMMISION_PCT'
  ```

  This argument is optional. If it is not specified, only the `AUDIT_CONDITION` argument determines whether an audit event must occur.

- **Object:** The object is the table or view that is being audited. It is passed as two arguments:
  - The schema that contains the object
  - The name of the object

  The example in the slide audits the `hr.employees` table by using the following arguments:

  ```
  object_schema => 'hr'
  object_name => 'employees'
  ```

- **Handler:** An optional event handler is a PL/SQL procedure that defines additional actions that must be taken during auditing. For example, the event handler can send an alert page to the administrator. If it is not defined, an audit event entry is inserted into the audit trail. If an audit event handler is defined, the audit entry is inserted into the audit trail and the audit event handler is executed.

  The audit event entry includes the FGA policy that caused the event, the user executing the SQL statement, and the SQL statement and its bind variables.

  The event handler is passed as two arguments:
  - The schema that contains the PL/SQL program unit
  - The name of the PL/SQL program unit

  The example in the slide executes the `SECURE.LOG_EMPS_SALARY` procedure by using the following arguments:

  ```
  handler_schema => 'secure'
  handler_module => 'log_emps_salary'
  ```

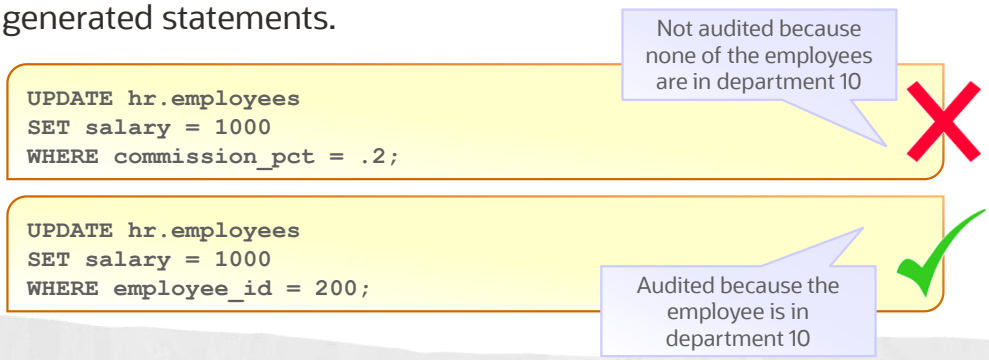
  By default, the audit trail always writes the SQL text and SQL bind information to LOBs. The default can be changed (for example, if the system would suffer performance degradation).

- **Status:** The status indicates whether the FGA policy is enabled. In the example in the slide, the following argument enables the policy:

  ```
  enable => TRUE
  ```

Audited DML Statements: Considerations

- Records are audited if the FGA predicate is satisfied and the relevant columns are referenced.
- DELETE statements are audited regardless of columns specified.
- MERGE statements are audited with the underlying INSERT, UPDATE, and DELETE generated statements.



With an FGA policy defined for DML statements, a DML statement is audited if the data rows (both new and old) that are being manipulated meet the policy predicate criteria.

However, if relevant columns are also specified in the policy definition, the statement is audited when the data meets the FGA policy predicate and the statement references the relevant columns defined.

For DELETE statements, specifying relevant columns during policy definition is not useful because all columns in a table are touched by a DELETE statement. Therefore, a DELETE statement is always audited regardless of the relevant columns.

MERGE statements are supported by FGA. The underlying INSERT, UPDATE, and DELETE statements are audited if they meet the defined INSERT, UPDATE, or DELETE FGA policies.

Using the previously defined FGA policy, the first statement is not audited whereas the second one is. None of the employees in department 10 receive a commission, but EMPLOYEE_ID=200 specifies an employee in department 10.

FGA Guidelines

- To audit all rows, use a `null` audit condition.
- To audit all columns, use a `null` audit column.
- Policy names must be unique.
- The audited table or view must already exist when you create the policy.
- If the audit condition syntax is invalid, an `ORA-28112` error is raised when the audited object is accessed.
- If the audited column does not exist in the table, no rows are audited.
- If the event handler does not exist, no error is returned and the audit record is still created.

For `SELECT` statements, FGA captures the statement itself and not the actual rows. However, when FGA is combined with Flashback Query, the rows can be reconstructed as they existed at that point in time.

For more details about the `DBMS_FGA` package, see the *Oracle Database PL/SQL Packages and Types Reference*.

Archiving and Purging the Audit Trail

- Periodically archive and purge the audit trail to prevent it from growing too large.
- Create an archive by:
  - Copying audit trail records to a database table
  - Using Oracle Audit Vault and Database Firewall
- Purge the audit trail by:
  - Creating and scheduling a purge job to run at a specified time by using the `DBMS_AUDIT_MGMT.CREATE_PURGE_JOB` PL/SQL procedure
  - Manually using the `DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL` PL/SQL procedure

Basic maintenance of the audit trail includes reviewing audit records and removing older records. The audit trail can grow to fill the available storage if not reviewed periodically and archived and purged. If the database audit trail fills the tablespace, audited actions do not complete.

Purging Audit Trail Records

- Schedule an automatic purge job:

```
DBMS_AUDIT_MGMT.CREATE_PURGE_JOB
(AUDIT_TRAIL_TYPE=> DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED,
 AUDIT_TRAIL_PURGE_INTERVAL => 12,
 AUDIT_TRAIL_PURGE_NAME => 'Audit_Trail_PJ',
 USE_LAST_ARCH_TIMESTAMP => TRUE,
 CONTAINER => DBMS_AUDIT_MGMT.CONTAINER_CURRENT);
```

- Manually purge the audit records:

```
DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL(
   AUDIT_TRAIL_TYPE   => DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED)
```

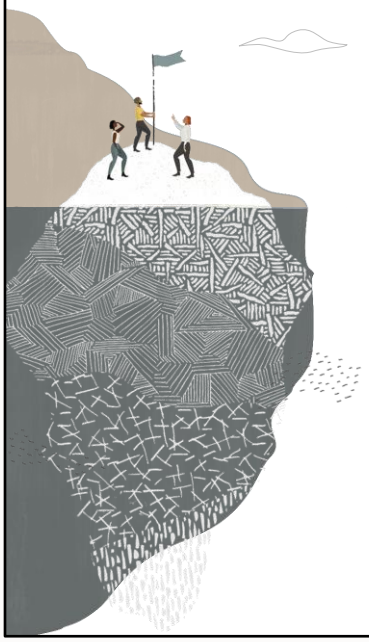
To perform audit trail purge tasks, you use the DBMS_AUDIT_MGMT package. You must have the AUDIT_ADMIN role to use the package. By mandate, Oracle Database audits all executions of DBMS_AUDIT_MGMT package procedures.

The DBMS_AUDIT_MGMT.CREATE_PURGE_JOB procedure includes the CONTAINER attribute. Setting CONTAINER to CONTAINER_CURRENT deletes audit trail records for the current pluggable database. Setting CONTAINER to CONTAINER_ALL deletes audit trail records for all pluggable databases, creating a job in the root, and the invocation of this job will invoke cleanup in all the PDBs.

USE_LAST_ARCH_TIMESTAMP specifies whether the last archived time stamp should be used for determining the records that should be deleted. Setting USE_LAST_ARCH_TIMESTAMP to TRUE indicates that only audit records created before the last archive time stamp should be deleted. A value of FALSE indicates that all audit records should be deleted. The default value is TRUE. Oracle recommends using this value because it helps guard against inadvertent deletion of records.

You can automate the cleanup process by creating and scheduling a cleanup purge job, or you can manually run a cleanup purge job. If you manually run cleanup purge jobs, use the DBMS_AUDIT_MGMT.CLEAN_AUDIT_TRAIL procedure with a new type value of DBMS_AUDIT_MGMT.AUDIT_TRAIL_UNIFIED.

Summary



- Describe DBA responsibilities for security and auditing
- Enable unified auditing
- Create unified audit policies
- Maintain the audit trail

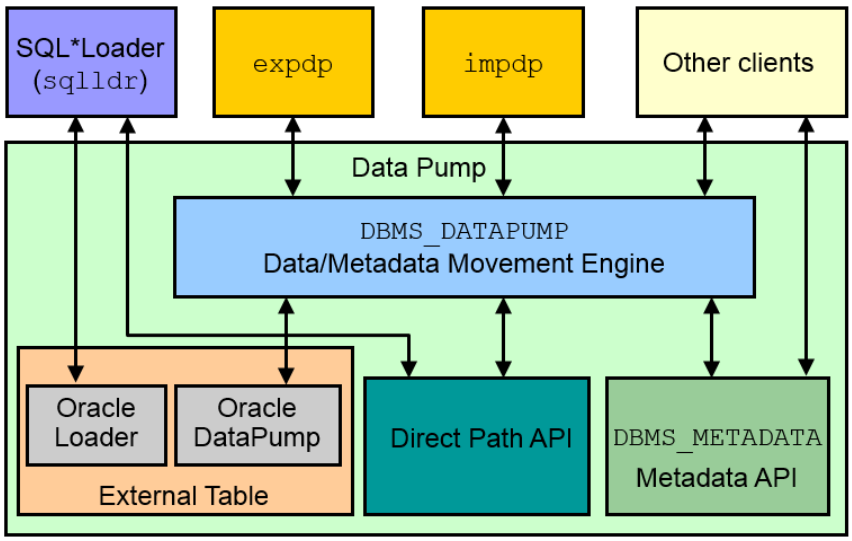
ORACLE
University

Introduction to Loading and Transporting Data

Objectives

- Describe ways to move data
- Explain the general architecture of Oracle Data Pump and SQL*Loader

Moving Data: General Architecture



Major functional components include:

- `DBMS_DATAPUMP`: It contains the API for high-speed export and import utilities for bulk data and metadata movement.
- Direct Path API (DPAPI): Oracle Database supports a Direct Path API interface that minimizes data conversion and parsing at both unload and load time.
- `DBMS_METADATA`: Used by worker processes for all metadata unloading and loading. Database object definitions are stored by using XML rather than SQL.
- External Table: With the `ORACLE_DATAPUMP` and `ORACLE_LOADER` access drivers, you can store data in external tables (that is, in platform-independent files). The `SELECT` statement reads external tables as though they were stored in an Oracle database.
- SQL*Loader: It has been integrated with external tables, providing automatic migration of loader control files to external table access parameters.
- `expdp` and `impdp`: They are thin layers that make calls to the `DBMS_DATAPUMP` package to initiate and monitor Data Pump operations.
- Other clients: Applications (such as replication, transportable tablespaces, and user applications) that benefit from this infrastructure. SQL*Plus may also be used as a client of `DBMS_DATAPUMP` for simple status queries against ongoing operations.

Oracle Data Pump: Overview

- As a server-based facility for high-speed data and metadata movement, Oracle Data Pump:
  - Can be called via `DBMS_DATAPUMP`
  - Provides the following tools:
    - `expdp` and `impdp`
    - GUI interface in Enterprise Manager Cloud Control
  - Provides several data movement methods:
    - Conventional path load
    - Direct path
    - External tables
    - Transportable tablespace
    - Network link support
  - Detaches from and reattaches to long-running jobs
  - Restarts Data Pump jobs

Oracle Data Pump enables very high-speed data and metadata loading and unloading of Oracle databases. The Data Pump infrastructure is callable via the `DBMS_DATAPUMP` PL/SQL package. Thus, custom data movement utilities can be built by using Data Pump.

Oracle Database provides the following tools:

- Command-line export and import clients called `expdp` and `impdp`, respectively
- An export and import interface in Enterprise Manager Cloud Control

Data Pump automatically decides the data access methods to use; these can be either direct path or external tables. Data Pump uses direct path load and unload when a table's structure allows it and when maximum single-stream performance is desired. However, if there are clustered tables, encrypted columns, or several other items, Data Pump uses external tables rather than direct path to move the data.

Conventional Path Load is used when Data Pump is not able to load data into a table by using either direct path or external tables. The ability to detach from and reattach to long-running jobs without affecting the job itself enables you to monitor jobs from multiple locations while they are running. All stopped Data Pump jobs can be restarted without loss of data as long as the metadata remains undisturbed. It does not matter whether the job is stopped voluntarily or involuntarily due to a crash.

Oracle Data Pump: Benefits

- Data Pump offers many benefits and features, such as:
  - Fine-grained object and data selection
  - Explicit specification of database version
  - Parallel execution
  - Network mode in a distributed environment
  - Remapping capabilities

The `EXCLUDE`, `INCLUDE`, and `CONTENT` parameters are used for fine-grained object and data selection.

You can specify the database version for objects to be moved (using the `VERSION` parameter) to create a dump file set that is compatible with a previous release of Oracle Database that supports Data Pump.

You can use the `PARALLEL` parameter to specify the maximum number of threads of active execution servers operating on behalf of the export job.

Network mode enables you to export from a remote database directly to a dump file set. This can be done by using a database link to the source system.

During import, you can change the target data file names, schemas, and tablespaces:

- Rename tables during an import operation.
- Remap data as it is being imported into a new database.

Oracle Data Pump: Benefits

- Data Pump offers many benefits and features, such as:
  - Data sampling and metadata compression
  - Compression of data during a Data Pump export
  - Security through encryption
  - Ability to export XMLType data as `CLOBs`

In addition, you can specify a percentage of data to be sampled and unloaded from the source database when performing a Data Pump export. This can be done by specifying the `SAMPLE` parameter.

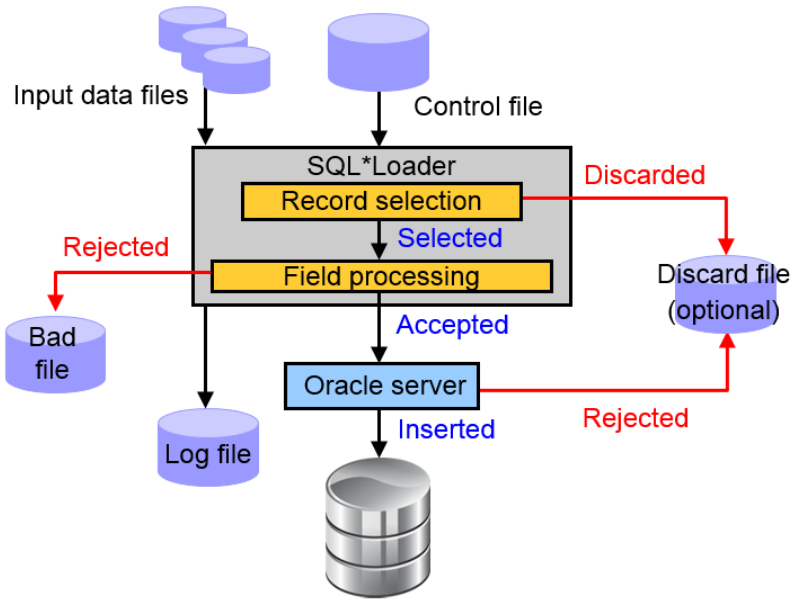
You can use the `COMPRESSION` parameter to indicate whether the metadata should be compressed in the export dump file so that it consumes less disk space. If you compress the metadata, it is automatically uncompressed during import. You can choose to compress both data and metadata, only data, only metadata, or no data during an export. This feature requires the Oracle Advanced Compression option.

You can also specify encryption options to encrypt. Encryption requires the Oracle Advanced Security option:

- Both data and metadata, only data, only metadata, no data, or only encrypted columns during an export
- A particular encryption algorithm to use during an export
- The type of security to use for performing encryption and decryption during an export. For example, perhaps the dump file set will be imported into a different or remote database and must remain secure in transit. Or perhaps the dump file set will be imported on-site using the Oracle Encryption Wallet, but it may also need to be imported off-site where the Oracle Encryption Wallet is not available.

You can also define that XMLType columns are to be exported in uncompressed `CLOB` format regardless of the XMLType storage format that was defined for them.

SQL Loader: Overview



SQL*Loader loads data from external files into the tables of an Oracle database. It has a powerful data parsing engine that puts little limitation on the format of the data in the data file.

SQL*Loader uses the following files:

Input data files: SQL*Loader reads data from one or more files (or operating system equivalents of files) that are specified in the control file. From SQL*Loader's perspective, the data in the data file is organized as records. A particular data file can be in fixed record format, variable record format, or stream record format. The record format can be specified in the control file with the `INFILE` parameter. If no record format is specified, the default is stream record format.

Control file: The control file is a text file that is written in a language that SQL*Loader understands. The control file indicates to SQL*Loader where to find the data, how to parse and interpret the data, where to insert the data, and so on. Although not precisely defined, a control file can be said to have three sections.

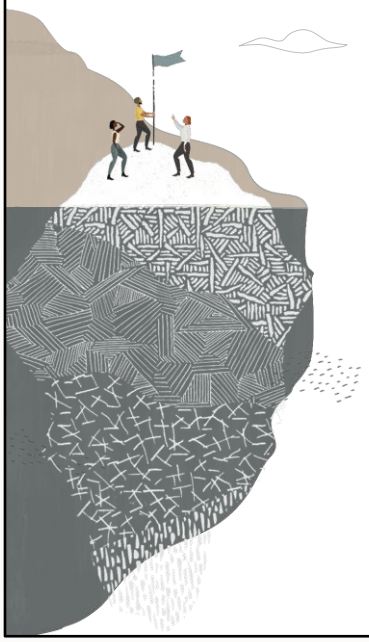
- The first section contains such sessionwide information as:
  - Global options, such as the input data file name and records to be skipped
  - `INFILE` clauses to specify where the input data is located
  - Data to be loaded
- The second section consists of one or more `INTO TABLE` blocks. Each of these blocks contains information about the table (such as the table name and the columns of the table) into which the data is to be loaded.
- The third section is optional and, if present, contains input data.

Log file: When SQL*Loader begins execution, it creates a log file. If it cannot create a log file, execution terminates. The log file contains a detailed summary of the load, including a description of any errors that occurred during the load.

Discard file: This file is created only when it is needed and only if you have specified that a discard file should be enabled. The discard file contains records that are filtered out of the load because they do not match any record-selection criteria specified in the control file.

Bad file: The bad file contains records that are rejected, either by SQL*Loader or by the Oracle database. Data file records are rejected by SQL*Loader when the input format is invalid. After a data file record is accepted for processing by SQL*Loader, it is sent to the Oracle database for insertion into a table as a row. If the Oracle database determines that the row is valid, the row is inserted into the table. If the row is determined to be invalid, the record is rejected, and SQL*Loader puts it in the bad file.

Summary



- Describe ways to move data
- Explain the general architecture of Oracle Data Pump and SQL*Loader

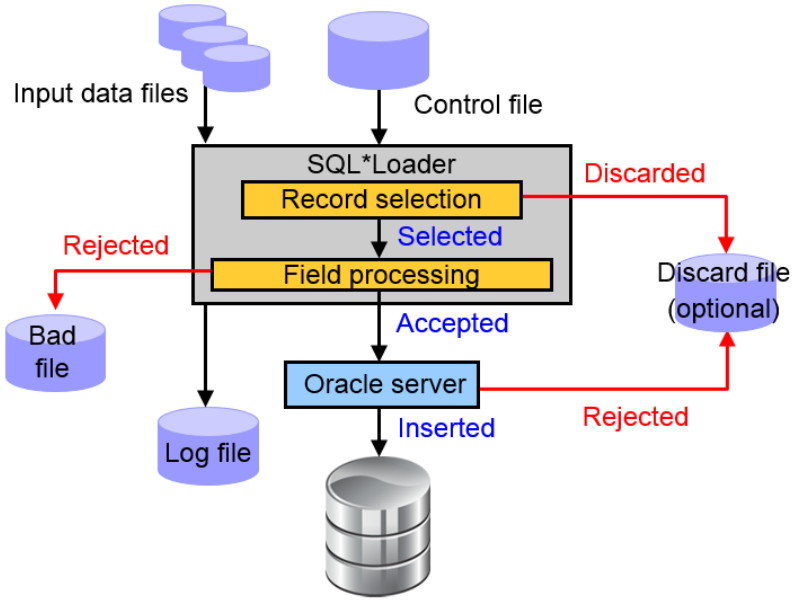
ORACLE
University

Loading Data

Objectives

Use SQL*Loader to load data from a non-Oracle database (or user files)

SQL Loader: Review



SQL*Loader uses the following files:

- **Input data files:** SQL*Loader reads data from one or more files (or operating system equivalents of files) that are specified in the control file.
- **Control file:** The control file is a text file that is written in a language that SQL*Loader understands.
- **Log file:** The log file contains a detailed summary of the load, including a description of any errors that occurred during the load.
- **Discard file:** The discard file contains records that are filtered out of the load because they do not match any record-selection criteria specified in the control file.
- **Bad file:** The bad file contains records that are rejected, either by SQL*Loader or by the Oracle database server.

Creating the SQL*Loader Control File

- The SQL*Loader control file contains:
  - Location of the data to be loaded
  - Data format
  - Configuration details:
    - Memory management
    - Record rejection
    - Interrupted load handling details
  - Data manipulation details

```
1         -- This is a sample control file
2  LOAD DATA
3  INFILE 'SAMPLE.DAT'
4  BADFILE 'sample.bad'
5  DISCARDFILE 'sample.dsc'
6  APPEND
7  INTO TABLE emp
8  WHEN (57) = '. '
9  TRAILING NULLCOLS
10 (hiredate SYSDATE,
    deptno POSITION(1:2) INTEGER EXTERNAL(3)
    NULLIF deptno=BLANKS,
    …
    empno POSITION(45) INTEGER EXTERNAL
    TERMINATED BY WHITESPACE,
    …
   )
```

The SQL*Loader control file is a text file that contains data definition language (DDL) instructions. DDL is used to control the following aspects of a SQL*Loader session:

- Where SQL*Loader finds the data to load
- How SQL*Loader expects that data to be formatted
- How SQL*Loader is being configured (including memory management, selection and rejection criteria, interrupted load handling, and so on) as it loads the data
- How SQL*Loader manipulates the data being loaded

The example in the slide includes:

- `LOAD DATA`: This statement indicates that this is the beginning of a new data load. If you are continuing a load that was interrupted in progress, use the `CONTINUE LOAD DATA` statement.
- `INFILE`: This keyword specifies the name of a data file containing data that you want to load.
- `BADFILE`: This keyword specifies the name of a file into which rejected records are placed.
- `DISCARDFILE`: This keyword specifies the name of a file into which discarded records are placed.
- `APPEND`: This keyword is one of the options that you can use when loading data into a table that is not empty. To load data into a table that is empty, use the `INSERT` keyword.
- `INTO TABLE`: This keyword enables you to identify tables, fields, and data types. It defines the relationship between records in the data file and tables in the database.
- `WHEN`: This clause specifies one or more field conditions that each record must match before SQL*Loader loads data.

- `TRAILING NULLCOLS`: This clause prompts SQL*Loader to treat any relatively positioned columns that are not present in the record as null columns.
- Field list: This list provides information about column formats in the table that is being loaded.

SQL*Loader Loading Methods

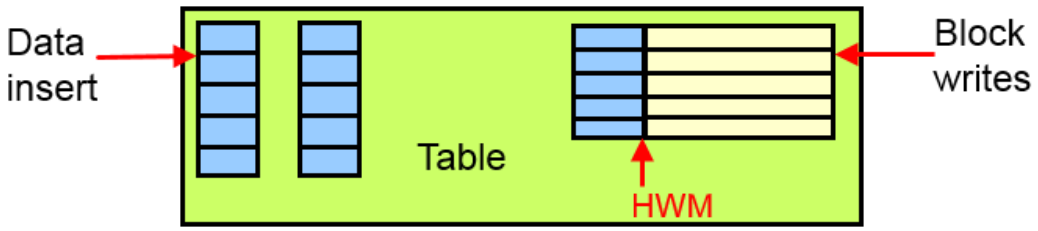
Conventional Load	Direct Path Load
Uses `COMMIT`	Uses data saves (faster operation)
Always generates redo entries	Generates redo only under specific conditions
Enforces all constraints	Enforces only `PRIMARY KEY`, `UNIQUE`, and `NOT NULL` constraints
Fires `INSERT` triggers	Does not fire `INSERT` triggers
Has the ability to load into clustered tables	Cannot load into clustered tables
Allows table modifications during load operation	Does not allow table modifications during load operation
Maintains index entries on each insert	Merges new index entries at the end of the load

You can use seven methods to load data with SQL*Loader.

- A conventional path load executes SQL `INSERT` statements to populate tables in an Oracle database. Conventional path loads use SQL processing and a database `COMMIT` operation for saving data. The insertion of an array of records is followed by a `COMMIT` operation. Each data load may involve several transactions.
- Direct path loads use data saves to format data blocks and write them directly to the data files bypassing the cache layer. This is why direct path loads are faster than conventional ones.

Protecting Against Data Loss

- Use data saves to protect against loss of data due to instance failure.
- Use the SQL*Loader `ROWS` parameter to specify when a data save should occur during a direct path load.



Use the `ROWS` parameter to specify the number of rows you want SQL*Loader to read from the input file before saving inserts in the database.

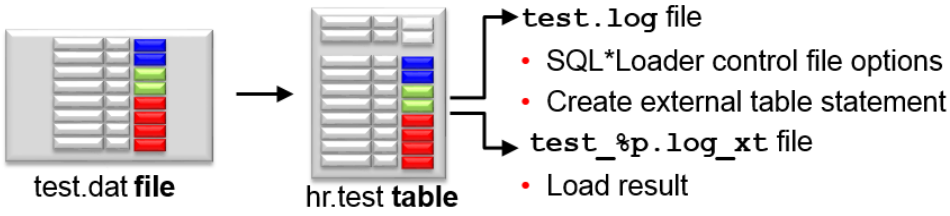
The intent of a data save is to provide an upper boundary (high-water mark) on the amount of work that is lost when an instance failure occurs during a long-running direct path load. Setting the value of `ROWS` to a small number adversely affects performance and data block space utilization.

The following features differentiate a data save from `COMMIT`:

- During a data save, only full database blocks are written to the database.
- The blocks are written after the high-water mark (HWM) of the table, as shown in the diagram in the slide.
- After a data save, the HWM is moved.
- Internal resources are not released after a data save.
- A data save does not end the transaction.
- If the `ROWS` parameter is specified, then SQL*Loader issues a data save after that many rows are loaded.
- Indexes are not updated at each data save. At the beginning of a direct path load, all indexes on the table are marked unusable. The indexes are updated at the end of the load. If the load is aborted after the indexes were marked unusable, then they will remain unusable.
- The control file option that lets the direct path API handle check constraints is `EVALUATE CHECK CONSTRAINTS`.

SQL*Loader Express Mode

- Specify a table name to initiate an express mode load.
- Table columns must be scalar data types (character, number, or datetime).
- A data file can contain only delimited character data.
- SQL*Loader uses table column definitions to determine input data types.
- There is no need to create a control file.



If you activate SQL*Loader express mode, specifying only the username and the `TABLE` parameter, it uses default settings for several other parameters. You can override most of the defaults by specifying additional parameters on the command line:

- `TERMINATED_BY` parameter, which specifies a field terminator
- `ENCLOSED_BY` parameter, which specifies a field enclosure character
- `OPTIONALLY_ENCLOSED_BY` parameter, which specifies an optional field enclosure character

The three delimiters can be a multicharacter string.

SQL*Loader express mode generates two files. The names of the log files come from the name of the table (by default).

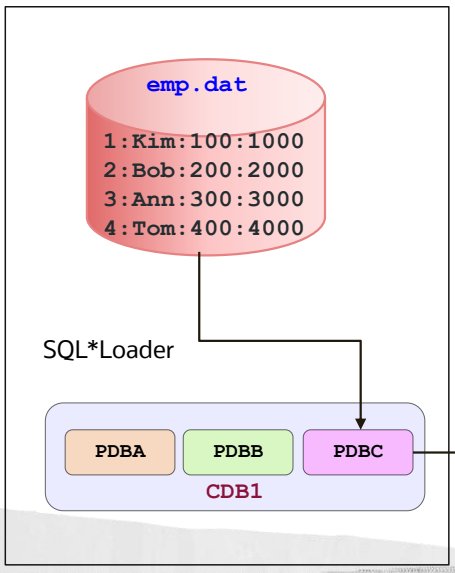
- A log file includes:
  - Control file output
  - SQL script for creating the external table and performing the load by using a SQL `INSERT /*+ APPEND */ AS SELECT` statement

Neither the control file nor the SQL script are used by SQL*Loader express mode. They are made available to you in case you want to use them as a starting point to perform operations using regular SQL*Loader or stand-alone external tables.

You can specify that direct path load be used instead of external tables with the `DIRECT=YES` parameter. You can also specify that a conventional path be used instead of external tables with `DIRECT=NO`.

- A log file similar to a SQL*Loader log file that describes the result of the operation. "`%p`" represents the process ID of the SQL*Loader process.

Using SQL*Loader to Load a Table in a PDB

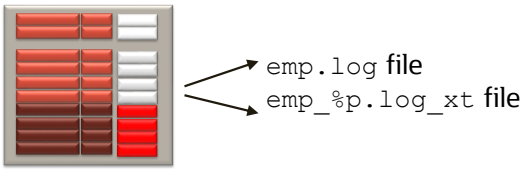


1. Use SQL*Loader express mode to insert rows into `HR.EMP` in `PDBC`.

```
$ sqlldr system@PDBC TABLE=hr.emp
```

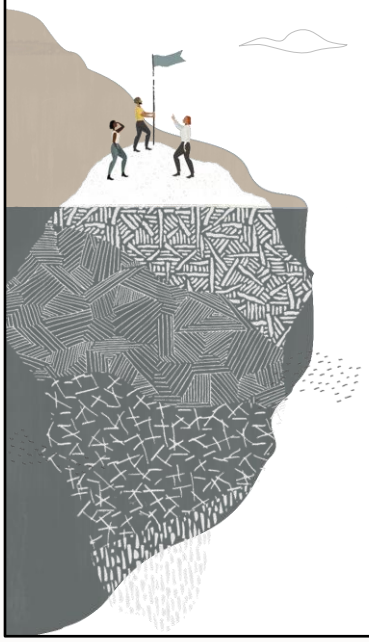
   - No need to prepare a control file.
   - The table columns must be scalar data types (character, number, or datetime).
   - SQL*Loader uses table column definitions to determine input data types.

2. Use log files to verify load operation.



You can use SQL*Loader express mode with PDBs as shown in the slide.

Summary



Use SQL*Loader to load data from a non-Oracle database (or user files)

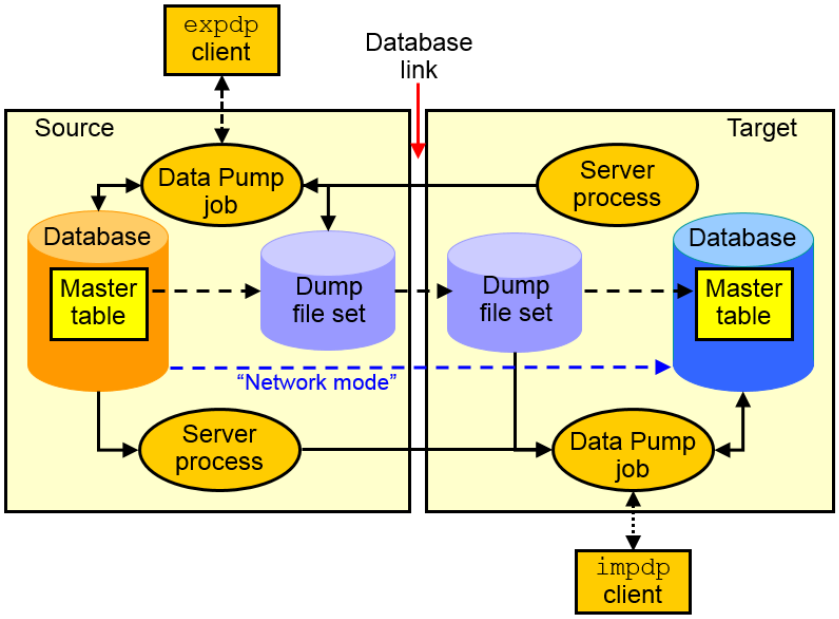
ORACLE
University

Transporting Data

Objectives

- Explain the general architecture of Oracle Data Pump
- Use Data Pump Export and Import to move data between Oracle databases
- Transport tablespaces between databases by using image copies or backup sets

Data Pump Export and Import Clients



Data Pump Export is a utility for unloading data and metadata into a set of operating system files called dump file sets. Data Pump Import is used to load metadata and data stored in an export dump file set into a target system.

The Data Pump API accesses its files on the server rather than on the client.

These utilities can also be used to export from a remote database directly to a dump file set or load the target database directly from a source database with no intervening files. This is known as network mode. This mode is particularly useful to export data from a read-only source database.

At the center of every Data Pump operation is the master table, which is a table created in the schema of the user running the Data Pump job. The master table maintains all aspects of the job. The master table is built during a file-based export job and is written to the dump file set as the last step. Conversely, loading the master table into the current user's schema is the first step of a file-based import operation and is used to sequence the creation of all objects imported.

Note: The master table is the key to Data Pump's restart capability in the event of a planned or unplanned stopping of the job. The master table is dropped when the Data Pump job finishes normally.

Data Pump Interfaces and Modes

- Data Pump Export and Import interfaces:
  - Command line
  - Parameter file
  - Interactive command line
  - Enterprise Manager Cloud Control
- Data Pump Export and Import modes:
  - Full
  - Schema
  - Table
  - Tablespace
  - Transportable tablespace
  - Transportable database

You can interact with Data Pump Export and Import by using one of the following interfaces:

- **Command-line interface:** Enables you to specify most of the export parameters directly on the command line
- **Parameter-file interface:** Enables you to specify all command-line parameters in a parameter file. The only exception is the `PARFILE` parameter.
- **Interactive-command interface:** Stops logging in to the terminal and displays the export or import prompts, where you can enter various commands. This mode is enabled by pressing Ctrl + C during an export operation that is started with the command-line interface or the parameter-file interface. Interactive-command mode is also enabled when you attach to an executing or stopped job.
- **Oracle Enterprise Manager Cloud Control:** Select Schema > Database Export/Import. In the menu, select the export or import operation you want to execute.

Data Pump Export and Import provide different modes for unloading and loading different portions of the database. The mode is specified on the command line by using the appropriate parameter. The available modes are also listed in the diagram in the slide.

- `FULL=YES`: All data and metadata of the CDB are to be exported. To perform a full export, you must have the `DATAPUMP_EXP_FULL_DATABASE` role.
- `SCHEMAS=hr,oe`: All data and metadata of the schemas are to be exported. This is the default mode for export. By default, if you do not have the `DATAPUMP_EXP_FULL_DATABASE` role, then only your own schema gets exported. If you have the `DATAPUMP_EXP_FULL_DATABASE` role, then you can specify a list of schemas.

- `TABLES=hr.employees, oe.sales`: Only the specified set of tables, partitions, and their dependent objects are unloaded.
- `TABLESPACES=tbs_app, tbs2`: Only the tables contained in a specified set of tablespaces are unloaded. If a table is unloaded, then its dependent objects are also unloaded. Both object metadata and data are unloaded.
- `TRANSPORT_TABLESPACES=tbs_app, tbs2`: Only object metadata contained in the tablespaces will be exported from the source database into the target database. The data is stored in data files. Because the data files do not get transported with the dump file, they should be copied to the target database before starting the import.
- `TRANSPORTABLE=ALWAYS` and `FULL=YES`: Both modes used together export all objects and data necessary to create a complete copy of the database. To import the full transportable database, use the `TRANSPORT_DATAFILES='datafile1','datafile2'` parameter to tell import that it is a transportable-mode import and from which data files to get the actual data.

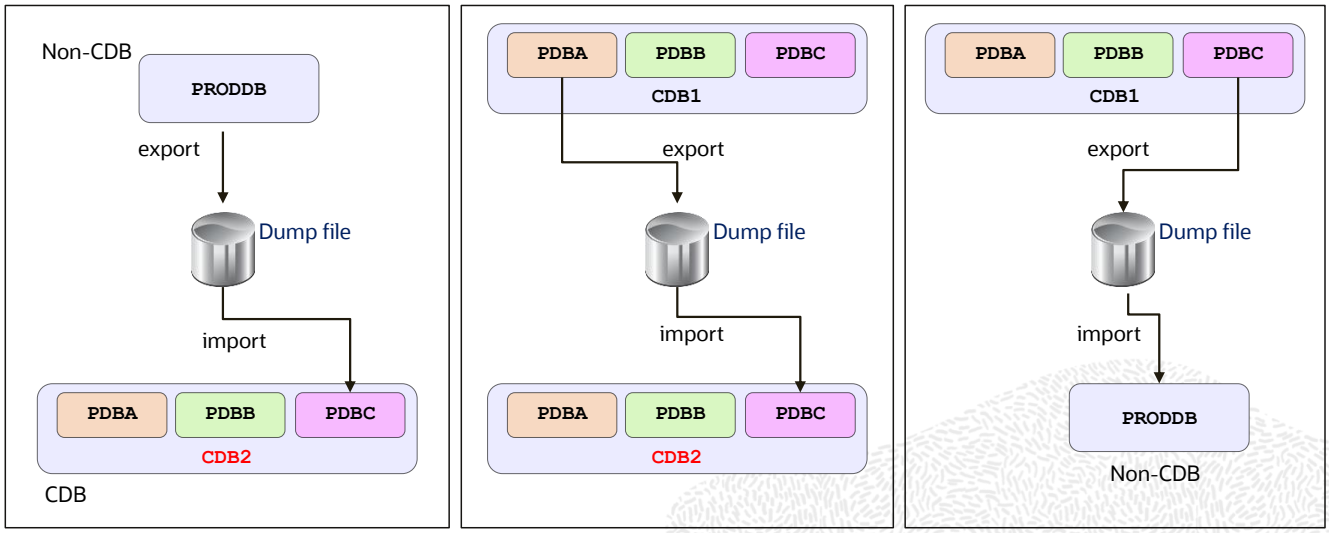
Data Pump Import Transformations

- You can remap:
  - Data files by using `REMAP_DATAFILE`
  - Tablespaces by using `REMAP_TABLESPACE`
  - Schemas by using `REMAP_SCHEMA`
  - Tables by using `REMAP_TABLE`
  - Data by using `REMAP_DATA`
  - Directory by using `REMAP_DIRECTORY`

Because object metadata is stored as XML in the dump file set, it is easy to apply transformations when DDL is being formed during import. Data Pump Import supports several transformations:

- `REMAP_DATAFILE` is useful when moving databases across platforms that have different file-system semantics. It changes the name of the source data file to the target data file name in all SQL statements where the source data file is referenced: `CREATE TABLESPACE`, `CREATE LIBRARY`, and `CREATE DIRECTORY`.
- `REMAP_TABLESPACE` enables objects to be moved from one tablespace to another.
- `REMAP_SCHEMA` provides the capability to change object ownership.
- `REMAP_TABLE` provides the ability to rename entire tables.
- `REMAP_DATA` provides the ability to remap data as it is being inserted.
- `REMAP_DIRECTORY` provides the ability to remap directories when you move databases between platforms.

Using Oracle Data Pump with PDBs



Use the PDB service name to export from or import into a PDB.

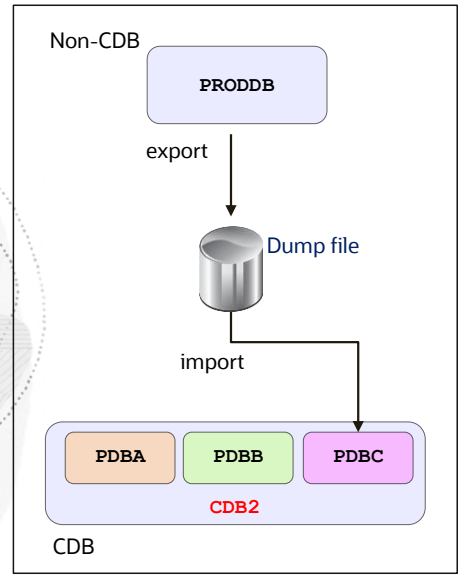
All types of export and import operations are possible between non-CDBs and PDBs. There are no supported CDB-wide Data Pump export or import operations, but only per-PDB Data Pump export or import operations, specifying the service name in the USERID clause.

- Export data from a non-CDB to import it into a PDB of a CDB.
- Export data from a PDB to import it into another PDB within the same CDB.
- Export data from a PDB to import it into a PDB of another CDB.
- Export data from a PDB to import it into a non-CDB.

Different types of Data Pump export and import are possible:

- Conventional export and import to export a full database (non-CDB or PDB) to import it into another database (non-CDB or PDB)
- Full transportable export and import to transport a full database (non-CDB or PDB) to import it into another database (non-CDB or PDB)
- Conventional or transportable tablespace export and import to export a full tablespace of a non-CDB or PDB to import it into another tablespace of a non-CDB or PDB
- Schema export and import to export a full schema of a non-CDB or PDB to import it into another tablespace of a non-CDB or PDB
- Table export and import to export a table of a non-CDB or PDB to import it into a non-CDB or PDB

Exporting from a Non-CDB and Importing into a PDB



1. Export PRODDB with the FULL=Y parameter:

```
$ expdp system@PRODDB FULL=Y DUMPFILE=proddb.dmp
```

2. If PDBC does not exist in CDB2, create PDBC in CDB2:

```
SQL> CONNECT sys@CDB1
SQL> CREATE PLUGGABLE DATABASE PDBC …;
```

3. Open PDBC.
4. Create a Data Pump directory in PDBC.
5. Copy the dump file to the Data Pump directory.
6. Create the same PRODDB tablespaces in PDBC for new local users' objects.
7. Import into PDBC with the FULL=Y and REMAP parameters:

```
$ impdp system@PDBC FULL=Y DUMPFILE=proddb.dmp
```

To export data from a non-CDB and import it into a PDB of a CDB, use the steps as described in the slide.

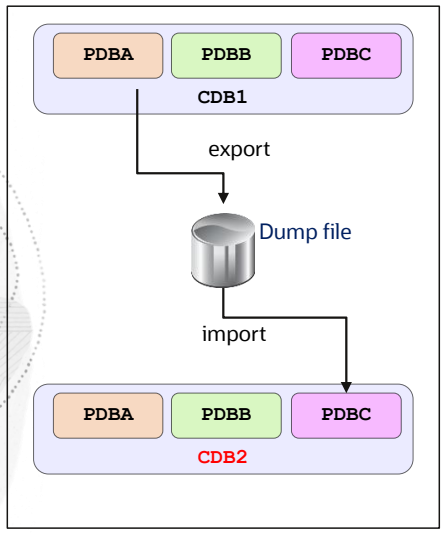
The choice made in the slide is to perform a conventional full database export from the non-CDB and a conventional full database import into the PDB. You can also perform a full transportable, tablespace, schema, or table-level export and import.

The tablespace export and import can be of either type: conventional or transportable.

The users exported from the non-CDB are re-created as local users in the PDB.

The tablespaces for the new local users and objects need to be created in the PDB before the import.

Exporting and Importing Between PDBs



1. Export PDBA from CDB1 with the FULL=Y parameter:

```
$ expdp system@PDBA FULL=Y …
```

2. If PDBC does not exist in CDB2, create PDBC in CDB2:

```
SQL> CONNECT sys@CDB2
SQL> CREATE PLUGGABLE DATABASE PDBC …;
```

3. Open PDBC.
4. Create a Data Pump directory in PDBC.
5. Copy the dump file to the Data Pump directory.
6. Create the same PDBA tablespaces in PDBC for new local users' objects.
7. Import into PDBC of CDB2 with the FULL and REMAP parameters:

```
$ impdp system@PDBC FULL=Y REMAP_SCHEMA=c##u:lu…
```

To export data from a PDB and import it into a PDB of the same or another CDB, use the steps as described in the slide.

The choice made in the slide is to perform a full database export from the PDB and a full database import into a PDB of another CDB. You can also perform a full transportable, tablespace, schema, or table-level export and import.

The tablespace export and import can be of either type: conventional or transportable.

The local users exported from the PDB are re-created as local users in the target PDB.

The common users are not re-created because their names prefixed with C## imply that a common user should be created. The statement fails with the following error message:

ORA-65094:invalid local user or role name

The only way to have common users re-created as local users is to use the clause REMAP_SCHEMA=C##xxx:local_user_name.

You can also create a common user in the CDB root.

Full Transportable Export/Import

- A full transportable export exports all objects and data necessary to create a complete copy of the database. Specify these parameter values:
  - TRANSPORTABLE=ALWAYS
    - FULL=Y

```
$ expdp user_name@pdb FULL=y DUMPFILE=expdat.dmp DIRECTORY=data_pump_dir
                         TRANSPORTABLE=always
```

- A full transportable import imports a dump file only if it has been created using the transportable option during export.
  - TRANSPORT_DATAFILES
  - If the TRANSPORTABLE parameter is specified, the NETWORK_LINK parameter is required.

Full Transportable Export

A full transportable export exports all objects and data necessary to create a complete copy of the database. A mix of data movement methods is used:

- Objects residing in transportable tablespaces have only their metadata unloaded into the dump file set. The data itself is moved when you copy the data files to the target database. The data files that must be copied are listed at the end of the log file for the export operation.
- Objects residing in nontransportable tablespaces (for example, SYSTEM and SYSAUX) have both their metadata and data unloaded into the dump file set, using direct path unload and external tables.

The example shows a full transportable export of a PDB.

Performing a full transportable export has the following restrictions:

- If the database being exported contains either encrypted tablespaces or tables with encrypted columns (either Transparent Data Encryption [TDE] columns or SecureFile LOB columns), then the ENCRYPTION_PASSWORD parameter must also be supplied.
- The source and target databases must be on platforms with the same endianness if there are encrypted tablespaces in the source database.
- If the source platform and the target platform are of different endianness, you must convert the data being transported so that it is in the format of the target platform. Use either the DBMS_FILE_TRANSFER package or the RMAN CONVERT command.
- A full transportable export is not restartable.

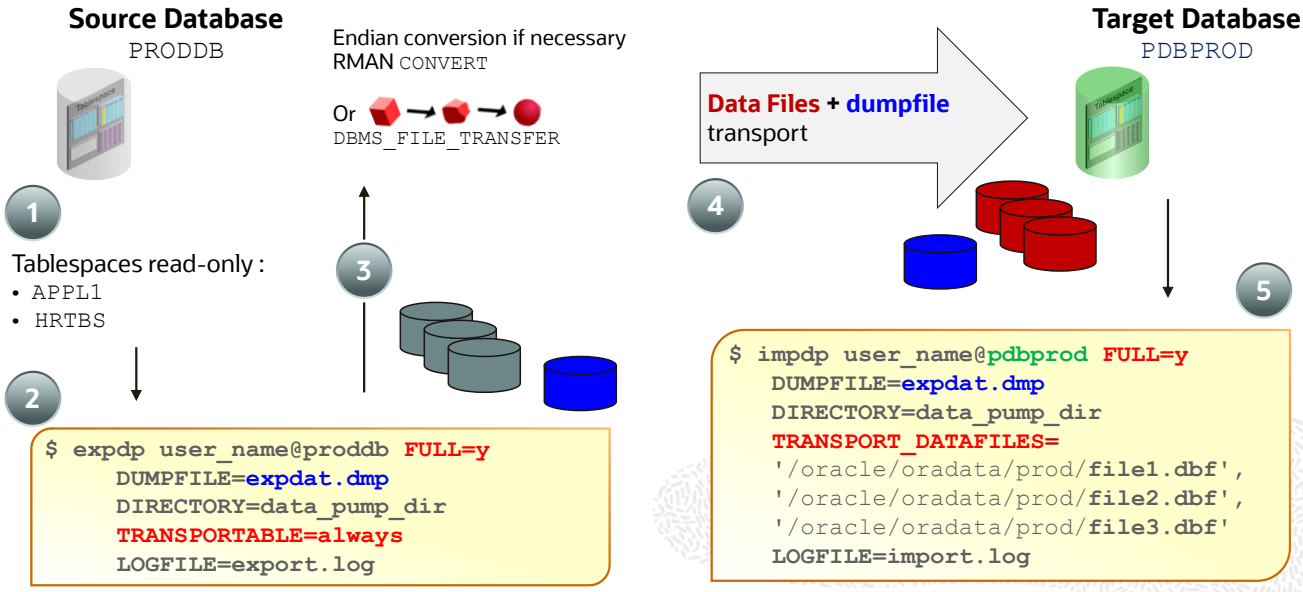
- All objects with storage that are selected for export must have all their storage segments either entirely within administrative, nontransportable tablespaces (SYSTEM / SYSAUX) or entirely within user-defined, transportable tablespaces. Storage for a single object cannot straddle the two kinds of tablespaces.
- When transporting a database over the network using full transportable export, tables with LONG or LONG RAW columns that reside in administrative tablespaces (such as SYSTEM or SYSAUX) are not supported.
- When transporting a database over the network using full transportable export, auditing cannot be enabled for tables stored in an administrative tablespace (such as SYSTEM and SYSAUX) if the audit trail information itself is stored in a user-defined tablespace.
- If the source platform and the target platform are of different endianness, then you must convert the data being transported so that it is in the format of the target platform. You can use the DBMS_FILE_TRANSFER package or the RMAN CONVERT command to convert the data.

Full Transportable Import

Performing a full transportable import has the following requirements:

- A full transportable import of encrypted tablespaces is not supported in network mode or dump file mode if the source and target platforms do not have the same endianess.
- When transporting a database over the network using full transportable import, tables with LONG or LONG RAW columns that reside in administrative tablespaces (such as SYSTEM or SYSAUX) are not supported.
- When transporting a database over the network using full transportable import, auditing cannot be enabled for tables stored in an administrative tablespace (such as SYSTEM and SYSAUX) if the audit trail information itself is stored in a user-defined tablespace.

Full Transportable Export/Import: Example



To perform a full transportable operation, perform the following steps:

1. Before the export, make all the user-defined tablespaces in the database read-only.
2. Invoke the Oracle Data Pump export utility as a user with the `DATAPUMP_EXP_FULL_DATABASE` role and specify the full transportable export options: `FULL=Y` and `TRANSPORTABLE=ALWAYS`. The `LOGFILE` parameter is important because it will contain the list of data files that need to be transported for the import operation.
3. Before the import, transport the dump file.
4. Transport the data files that you may have converted. If you are transporting the database to a platform different from the source platform, determine if cross-platform database transport is supported for both the source and target platforms. If both platforms have the same endianness, no conversion is necessary. Otherwise, you must do a conversion of each tablespace in the database either at the source or target database using either `DBMS_FILE_TRANSFER` or the RMAN `CONVERT` command.
5. Invoke the Oracle Data Pump import utility as a user with the `DATAPUMP_IMP_FULL_DATABASE` role and specify the full transportable import options: `FULL=Y` and `TRANSPORT_DATAFILES`.
6. Make the source tablespaces read-write. You can perform this operation before step 5.

Transporting a Database Over the Network: Example

- To transport a database over the network, perform an import using the `NETWORK_LINK` parameter.
  1. Create a database link in the target to the source database.
  2. Make the user-defined tablespaces in the source database read-only.
  3. Transport the data files for all the user-defined tablespaces from the source to the target location.
  4. Perform conversion of the data files if necessary.
  5. Import in the target database.

```
$ impdp username@pdbname full=Y network_link = sourcedb
      transportable = always
      transport_datafiles = '/oracle/oradata/prod/sales01.dbf',
                            '/oracle/oradata/prod/cust01.dbf'
      logfile=import.log
```

To transport a database over the network, use import with the `NETWORK_LINK` parameter. The import is performed using a database link, and there is no dump file involved.

If the source or target database is a PDB, use the PDB service name in the `USERID` clause.

The Oracle Data Pump network import copies the metadata for objects contained within the user-defined tablespaces and both the metadata and data for user-defined objects contained within the administrative tablespaces, such as `SYSTEM` and `SYSAUX`.

When the import is complete, the user-defined tablespaces are in read-write mode.

Make the user-defined tablespaces read-write again in the source database.

Using RMAN to Transport Data Across Platforms

- Transporting databases, data files, and tablespaces across platforms:
  - Cross-platform transport (with different endian formats)
  - Based on image copies and backup sets
  - Use of inconsistent tablespace backups
- Benefits:
  - Reduced down time for platform migrations
  - Choice of compression and multisection
  - Not cataloged in control file, not used for regular restore operations

RMAN enables you to transport databases, data files, and tablespaces across platforms. This includes transporting tablespaces across platforms with different endian formats (byte ordering). You can convert a database on the destination host or source host. For platforms that have the same endian format, no conversion is needed.

- Cross-platform data transport can be based on image copies or backup sets.
- You can also create cross-platform inconsistent tablespace backups by using image copies and backup sets. An inconsistent tablespace backup is one that is created when the tablespace is not in read-only mode.
- With the use of backup sets, you can choose compression and multisection options, which reduce the overall transport time.

Note: RMAN does not catalog backup sets created for cross-platform transport in the control file. This ensures that backup sets created for cross-platform transportation are not used during regular restore operations.

RMAN CONVERT Command

- RMAN:
  - Converts tablespaces, data files, or databases to the format of a destination platform
  - Does not change input files
  - Writes converted files to the output destination
  - Can convert on the source or destination platform
  - Assumes you initiate the data transfer

```
rman target sys@orcl
RMAN> ALTER TABLESPACE bartbs READ ONLY;

RMAN> CONVERT TABLESPACE bartbs
   TO PLATFORM 'Solaris Operating System (x86-64)'
   FORMAT '/tmp/transport/%U';
```

You use the RMAN `CONVERT` command to convert a tablespace, data file, or database to the format of a destination platform in preparation for transport across different platforms:

- `CONVERT DATAFILE`
- `CONVERT TABLESPACE`
- `CONVERT DATABASE`

Input files are not altered by `CONVERT` because the conversion is not performed in place. Instead, RMAN writes converted files to a specified output destination.

When you use the RMAN `CONVERT` command to convert data, you can convert the data either on the source platform after running Data Pump export or on the target platform before running Data Pump import. In either case, you must transfer the data files from the source system to the target system.

Restrictions: The `CONVERT` command does not process user data types that require endian conversions. To transport objects between databases built on underlying types that store data in a platform-specific format, use the Data Pump Import and Export utilities.

For detailed prerequisites, usage, restrictions, and syntax, see *Oracle Database Backup and Recovery Reference* and *Oracle Database Administrator's Guide*.

Transporting Data with Minimum Down Time

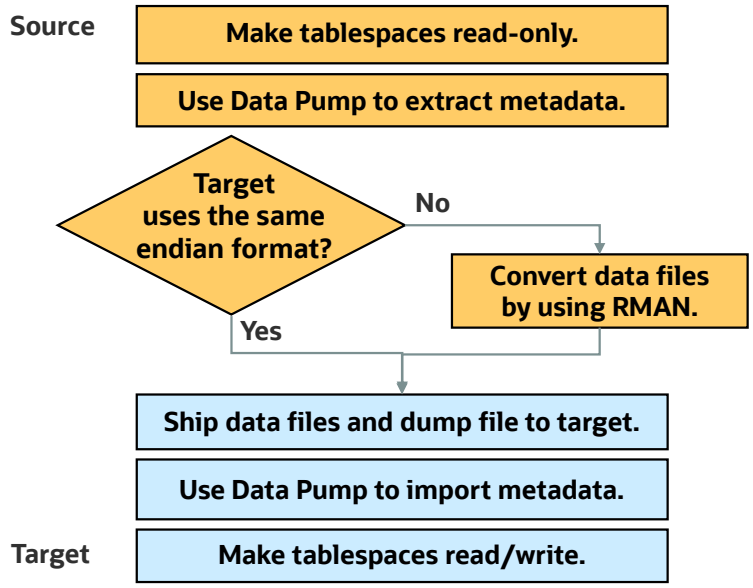
- Consider the required database open mode and endian format:
  - Database transport: READ ONLY, same endian format
  - Tablespace transport: READ WRITE, different endian format
- Example:
  1. Create a database incremental level 0 backup and apply it to the destination.
  2. Create incremental backups and apply them to the destination.
  3. Repeat: Create and apply incremental backups.
  4. Perform the final incremental backup in `READ ONLY` mode, apply it, and open both databases consistent with each other.

When you develop a database transport strategy, you need to consider the endian format of the platforms and the database open mode.

- Transport at the database level requires the same endian format (on source and destination) and `READ ONLY` mode of the source database (which is not desirable for a database that users need to update frequently).
- Tablespaces and backup sets can be transported across platforms of different endian format, while the source database remains online (in `READ WRITE` mode).

The slide shows a sample workflow that considers these requirements. It is a strategy to minimize down time by performing most of the work when the database is open in `READ WRITE` mode. The database is in `READ ONLY` mode only for the final step (a small incremental backup), which is required so that both databases can be opened in a completely consistent state.

Transporting a Tablespace by Using Image Copies



To transport a tablespace from one platform to another (source to target), data files belonging to the tablespace set must be converted to a format that can be understood by the target or destination database. Although with Oracle Database disk structures conform to a common format, it is possible for the source and target platforms to use different endian formats (byte ordering). When going to a different endian platform, you can use the `CONVERT` command of the RMAN utility to convert the byte ordering. This operation can be performed on either the source or the target platforms. For platforms that have the same endian format, no conversion is needed.

The graphic in the slide depicts the possible steps to transport tablespaces from a source platform to a target platform. However, it is possible to perform the conversion after shipping the files to the target platform. The last two steps must be executed on the target platform.

Basically, the procedure is the same as when using previous releases of Oracle Database server except when both platforms use different endian formats. It is assumed that both platforms are cross-transportable compliant.

Determining the Endian Format of a Platform

- Cross-platform transportable tablespaces:
  - Simplify moving data between data warehouse and data marts
  - Allow database migration from one platform to another
  - Allow the same character set on source and target platforms
- List of supported platforms and their endian formats:

```
SELECT * FROM V$TRANSPORTABLE_PLATFORM;
```

- Determine the endian format of source and target platforms:

```
SELECT tp.endian_format
FROM   v$transportable_platform tp, v$database sp
WHERE  tp.platform_name = sp.platform_name;
```

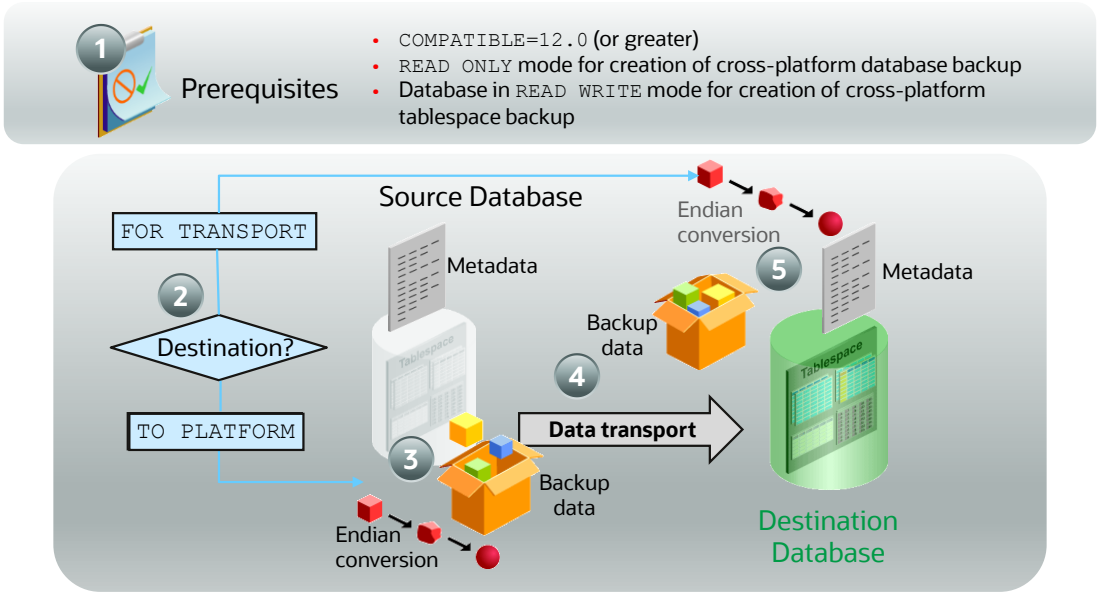
Using transportable tablespaces, Oracle data files (containing table data, indexes, and almost every other Oracle database object) can be directly transported from one database to another. Transportable tablespaces also provide a mechanism for transporting metadata.

You can use the transportable tablespace feature to move data across platform boundaries (with the same character set). This simplifies the distribution of data from a data warehouse environment to data marts, which often run on smaller platforms. It also allows a database to be migrated from one platform to another by rebuilding the dictionary and transporting the user tablespaces.

Moving data by using transportable tablespaces is much faster than performing either an export/import or an unload/load of the same data. This is because the data files containing all the actual data are just copied to the destination location, and you use Data Pump to transfer only the metadata of the tablespace objects to the new database.

To be able to transport data files from one platform to another, you must ensure that both the source system and the target system are running on one of the supported platforms. Query `V$TRANSPORTABLE_PLATFORM` to determine whether the endian ordering is the same on both platforms. `V$DATABASE` has two columns that can be used to determine your own (source) platform name and platform identifier.

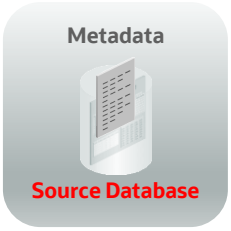
Transporting Data with Backup Sets



The graphic in the slide illustrates transporting data with backup sets:

1. Before you create a backup set that can be used for cross-platform data transportation, the following prerequisites must be met:
   - The `COMPATIBLE` parameter must be set to 12.0 or greater.
   - To transport an entire database, the source database must be open in read-only mode, because the `SYS` and `SYSAUX` tablespaces participate in the transport.
   - To transport tablespaces, when you use the `DATAPUMP` clause, the database must be open in read/write mode so that Data Pump can access the metadata.
2. There are two alternatives that affect the location of the endian conversion (if needed).
   - The `FOR TRANSPORT` clause indicates that the backup set can be transported to any destination database. If the destination database uses an endian format that is different from that of the source database, the endian format conversion is performed on the destination database.
   - The `TO PLATFORM` clause indicates that the conversion is performed on the source database.
   - In both cases, the `DATAPUMP` clause indicates that an export dump file for the tablespaces must be created. In this case, the database must be opened in read/write mode. The export can be performed after the last incremental backup.
3. A tablespace indicates metadata and backup data of the source database.
4. The gray arrow indicates the data transport.
5. Backup data and metadata and a tablespace are transported in the destination database.

Transporting a Tablespace



1. Verify the prerequisites.
2. Start an RMAN session in the source database.
3. Query the exact name of the destination platform.
4. Change the tablespace to read-only.

```
RMAN> ALTER TABLESPACE test READ ONLY;
```

5. Perform a cross-platform transportable backup and a Data Pump export.
   - Conversion on the destination host

```
RMAN> BACKUP FOR TRANSPORT FORMAT '/bkp/test.bck'
      DATAPUMP FORMAT '/bkp/test_meta.bck' TABLESPACE test;
```

   - Conversion on the source host

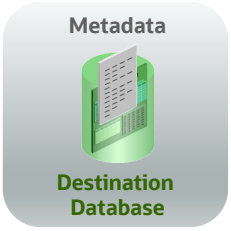
```
RMAN> BACKUP TO PLATFORM 'HP Tru64 UNIX'
      FORMAT '/bkp/test.bck'
      DATAPUMP FORMAT '/bkp/test_meta.bck' TABLESPACE test;
```

Perform the following steps to transport a tablespace:

1. Verify the prerequisites: The source database must be opened in read/write mode.
2. Start an RMAN session and connect to the target instance.
3. To transport tablespaces across platforms, query the exact name of the destination platform.
4. Change the tablespace to read-only.
5. Back up the source tablespace by using the `BACKUP` command with the `TO PLATFORM` or `FOR TRANSPORT` clause to indicate where the conversion takes place. Use the `DATAPUMP` clause to specify the location of a backup set that contains metadata of the named tablespace.

Note: The `ALLOW INCONSISTENT` clause of the `BACKUP` command enables you to back up tablespaces that are not in read-only mode. Although the backup is created, you cannot plug these tablespaces directly into the target database because they are inconsistent. You must later create an incremental backup of the tablespaces when they are in read-only mode. This incremental backup must contain the `DATAPUMP` clause that creates an export dump file of the tablespace metadata.

Transporting a Tablespace



6. Move the backup sets and the Data Pump export dump file to the destination host.
7. Connect to the destination host as TARGET.
8. Restore the cross-transportable backup and the Data Pump export.

```
RMAN> RESTORE FOREIGN TABLESPACE test
      FORMAT '/oracle/test.dbf'
      FROM BACKUPSET '/bkp/test.bck'
      DUMP FILE FROM BACKUPSET '/bkp/test_meta.bck' ;
```

6. Disconnect from the source database and move the backup sets and the Data Pump file to the destination host. You can use operating system utilities for this task.
7. Connect to the destination host, to which the tablespace is transported, as TARGET. Ensure that the destination database is opened in read-write mode.
8. Use the RESTORE command in the destination database as shown in the slide. The FOREIGN TABLESPACE clause points to the HP source data file. The FORMAT clause indicates the destination location. The DUMP FILE FROM BACKUPSET clause restores the required metadata from the Data Pump file.

An additional example:

1. Create cross-platform, inconsistent, incremental backups with the ALLOW INCONSISTENT clause:

```
BACKUP INCREMENTAL FROM SCN=2720649 FOR TRANSPORT
ALLOW INCONSISTENT FORMAT '/home/u_inc1.bkp' TABLESPACE
users;
```

2. Restore the inconsistent cross-platform tablespace backup with the RESTORE FOREIGN TABLESPACE command.
3. Recover the restored data files copied with cross-platform incremental backups with the RECOVER FOREIGN DATAFILECOPY command.

Transporting Inconsistent Tablespaces

- Create cross-platform inconsistent incremental backups with the `ALLOW INCONSISTENT` clause.
- Restore the inconsistent cross-platform tablespace backup with the `RESTORE FOREIGN TABLESPACE` command.
- Recover restored data file copies with cross-platform incremental backups with the `RECOVER FOREIGN DATAFILECOPY` command.

To use inconsistent tablespaces in a workflow like the one described on the previous pages:

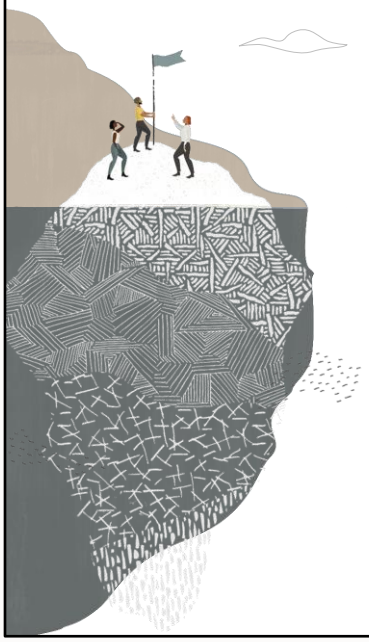
- Create a cross-platform inconsistent incremental backup with the `ALLOW INCONSISTENT` clause:

```
BACKUP INCREMENTAL FROM SCN=2720649 FOR TRANSPORT
ALLOW INCONSISTENT FORMAT '/u01/backup/inc1.bkp' TABLESPACE users;
```

- Restore the inconsistent cross-platform tablespace backup with the `RESTORE FOREIGN TABLESPACE` command.
- Recover restored data file copies with cross-platform incremental backups by using the `RECOVER FOREIGN DATAFILECOPY` command.

Note: The `ALLOW INCONSISTENT` clause enables you to back up tablespaces that are not in read-only mode. Although the backup is created, you cannot plug these tablespaces directly into the target database because they are inconsistent. You must later create an incremental backup of the tablespaces when they are in read-only mode. This incremental backup must contain the `DATAPUMP` clause that creates an export dump file of the tablespace metadata.

Summary



- Explain the general architecture of Oracle Data Pump
- Use Data Pump Export and Import to move data between Oracle databases
- Transport tablespaces between databases by using image copies or backup sets

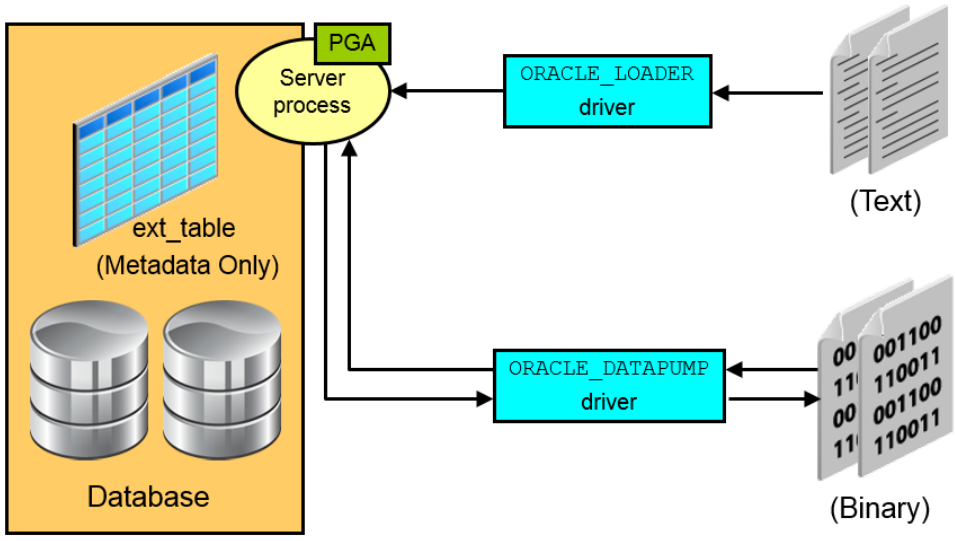
ORACLE
University

Using External Tables to Load and Transport Data

Objectives

Use external tables to move data via platform-independent files

External Tables



External tables allow SQL statements to access data in external sources as if it were in a table in the database. You can connect to the database instance and create metadata for the external table by using DDL. The DDL for an external table consists of two parts:

- One part that describes the Oracle Database column types
- Another part that describes the mapping of the external data to the Oracle Database data columns

An external table does not describe any data that is stored in the database. It also does not describe how data is stored in the external source. Instead, it describes how the external table layer must present the data to the server. It is the responsibility of the access driver and the external table layer to do the necessary transformations required on the data in the external file so that it matches the external table definition. External tables are read-only; therefore, no DML operations are possible.

There are two access drivers used with external tables.

- The `ORACLE_LOADER` access driver can be used only to read table data from an external table and load it into the database. It uses text files as the data source.
- The `ORACLE_DATAPUMP` access driver can both load table data from an external file into the database and also unload data from the database into an external file. It uses binary files as the external files.

You can partition data contained in external tables, which allows you to take advantage of the same performance improvements provided when you partition tables stored in a database.

External Tables: Benefits

- Data can be used directly from the external file or loaded into another database.
- External data can be queried and joined directly in parallel with tables residing in the database, without requiring it to be loaded first.
- The results of a complex query can be unloaded to an external file.
- You can combine generated files from different sources for loading purposes.

The data files created for the external table can be moved and used as the data files for another external table in the same database or a different database.

External data can be queried and joined directly in parallel to tables residing in the database, without requiring the data to be loaded first. You can choose to have your applications directly access external tables with the `SELECT` command, or you can choose to have data loaded first into a target database.

The results of a complex query can be unloaded to an external file by using the `ORACLE_DATAPUMP` access driver.

Data files that are populated by different external tables can all be specified in the `LOCATION` clause of another external table. This provides an easy way of aggregating data from multiple sources. The only restriction is that the metadata for all the external tables must be exactly the same.

ORACLE_LOADER Access Driver

```
CREATE TABLE extab_employees
   (employee_id       NUMBER(4),
    first_name        VARCHAR2(20),
    last_name         VARCHAR2(25),
    hire_date         DATE)
ORGANIZATION EXTERNAL
   (TYPE ORACLE_LOADER
    DEFAULT DIRECTORY extab_dat_dir
    ACCESS PARAMETERS
      (records delimited by newline
       badfile extab_bad_dir:'empxt%a_%p.bad'
       logfile extab_log_dir:'empxt%a_%p.log'
       fields terminated by ','
       missing field values are null
       (employee_id, first_name, last_name,
        hire_date char date_format date mask "dd-mon-yyyy"))
      LOCATION ('empxt1.dat', 'empxt2.dat'))
   PARALLEL REJECT LIMIT UNLIMITED;
```

The metadata for an external table is created by using the `CREATE TABLE` statement. The `ORACLE_LOADER` access driver uses the SQL*Loader syntax to define the external table. This command does not create the external text files.

The example in the slide shows three directory objects (`EXTAB_DAT_DIR`, `EXTAB_BAD_DIR`, and `EXTAB_LOG_DIR`) that are created and mapped to existing OS directories to which the user is granted access.

When the `EXTAB_EMPLOYEES` table is accessed, SQL*Loader functionality is used to load the table, and at that instance, the log file and bad file are created.

Best-practice tip: If you have a lot of data to load, enable `PARALLEL` for the load operation:

```
ALTER SESSION ENABLE PARALLEL DML;
```

ORACLE_DATAPUMP Access Driver

```
CREATE TABLE ext_emp_query_results
   (first_name, last_name, department_name)
ORGANIZATION EXTERNAL
  (
    TYPE ORACLE_DATAPUMP
    DEFAULT DIRECTORY ext_dir
    LOCATION ('emp1.exp','emp2.exp','emp3.exp')
  )
PARALLEL
AS
SELECT e.first_name,e.last_name,d.department_name
FROM   employees e, departments d
WHERE  e.department_id = d.department_id AND
       d.department_name in ('Marketing', 'Purchasing');
```

This example shows you how the external table population operation can help to export a selective set of records resulting from the join of the `EMPLOYEES` and `DEPARTMENTS` tables.

Because the external table can be large, you can use a parallel populate operation to unload your data to an external table. As opposed to a parallel query from an external table, the degree of parallelism of a parallel populate operation is constrained by the number of concurrent files that can be written to by the access driver. There is never more than one parallel execution server writing into one file at a particular point in time.

The number of files in the `LOCATION` clause must match the specified degree of parallelism because each input/output (I/O) server process requires its own file. Any extra files that are specified are ignored. If there are not enough files for the specified degree of parallelism, the degree of parallelization is lowered to match the number of files in the `LOCATION` clause.

The external table is read-only after it has been populated. The `SELECT` command can be very complex, allowing specific information to be populated in the external table. The external table, having the same file structure as binary Data Pump files, can then be migrated to another system, and imported with the `impdp` utility or read as an external table.

Note: For more information about the `ORACLE_DATAPUMP` access driver parameters, see the *Oracle Database Utilities* guide.

External Tables

- Querying an external table:

```
SELECT * FROM extab_employees;
```

- Querying and joining an external table with an internal table:

```
SELECT e.employee_id, e.first_name, e.last_name, d.department_name
FROM departments d, extab_employees e
WHERE d.department_id = e.department_id;
```

- Appending data from an external table to an internal table:

```
INSERT /*+ APPEND */ INTO hr.employees
SELECT * FROM extab_employees;
```

External tables are queried just like internal database tables. The first example illustrates querying an external table named `EXTAB_EMPLOYEES` and only displaying the results. The results are not stored in the database.

The second example shows the joining of an internal table named `DEPARTMENTS` with an external table named `EXTAB_EMPLOYEES` and only displaying the results.

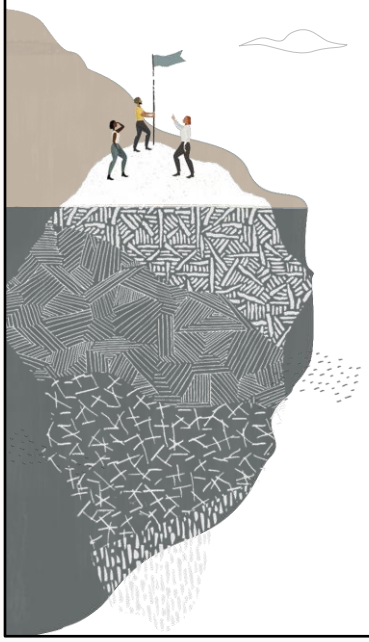
The third example in the slide illustrates the direct appending of an internal table data with the query and loading of data from an external table.

Viewing Information About External Tables

Dictionary Views	Description
[DBA| ALL| USER]_EXTERNAL_TABLES	Specific attributes
[DBA| ALL| USER]_EXTERNAL_LOCATIONS	Data sources
[DBA| ALL| USER]_TABLES	All tables
[DBA| ALL| USER]_TAB_COLUMNS	Columns of tables
[DBA| ALL]_DIRECTORIES	Directory objects

The data dictionary views listed in the slide provide information about external tables. See *Oracle Database Reference* for detailed information about each view.

Summary



Use external tables to move data via platform-independent files

Practice Overview

- Querying External Tables
- Unloading External Tables

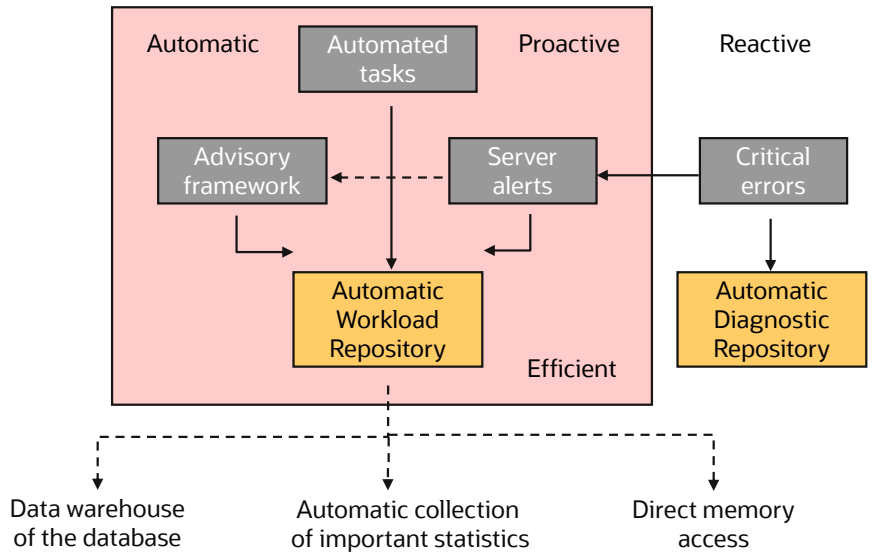
ORACLE
University

Automated Maintenance Tasks: Overview

Objectives

- Describe Oracle Database's proactive database maintenance infrastructure
- Discuss automated maintenance tasks
- Explain maintenance windows

Proactive Database Maintenance Infrastructure



Proactive database maintenance is made easy by the sophisticated infrastructure of Oracle Database, which includes the following main elements:

- The Automatic Workload Repository (AWR) is a built-in repository in each Oracle database. At regular intervals, the Oracle Database server takes a snapshot of all its vital statistics and workload information, and stores this data in the AWR. The captured data can be analyzed by you, by the database server itself, or by both.
- Using automated tasks, the database server performs routine maintenance operations, such as regular backups, refreshing optimizer statistics, and database health checks.

Reactive database maintenance includes critical errors and conditions discovered by database health checkers:

- For problems that cannot be resolved automatically and require administrators to be notified (such as running out of space), the Oracle Database server provides server-generated alerts. The Oracle Database server, by default, monitors itself and sends out alerts to notify you of problems. The alerts notify you and often also provide recommendations on how to resolve the reported problem. The DBA can also be alerted by users whose transactions are locked by other users' transactions and are waiting for locks to be released.
- Recommendations are generated from several advisors, each of whom is responsible for a subsystem. For example, there are memory, segment, and SQL advisors.

This lesson focuses on automated maintenance tasks.

Automated Maintenance Tasks: Components

- Automated maintenance tasks
  - Tasks started automatically
  - Performs maintenance operations on the database
  - Controlled by Maintenance windows
- Maintenance windows
  - Predefined time intervals
  - Intended to occur during a period of low system load

Automated maintenance tasks are tasks that are started automatically at regular intervals to perform maintenance operations on the database. An example is a task that gathers statistics on schema objects for the query optimizer.

Automated maintenance tasks are executed when the system load is expected to be light. You can enable and disable individual maintenance tasks, and configure when these tasks run and what resource allocations they are allotted.

Automated maintenance tasks run in maintenance windows, which are predefined time intervals that are intended to occur during a period of low system load. You can customize maintenance windows based on the resource usage patterns of your database, or disable certain default windows from running. You can also create your own maintenance windows.

Automated Maintenance Tasks: Components

- Oracle Scheduler
  - An enterprise job scheduler
  - A job is created for each maintenance task
  - Jobs are scheduled to run in a maintenance window when it opens.
- Oracle Database Resource Manager
  - Manage resource allocation for a database
  - Uses the `DEFAULT_MAINTENANCE_PLAN` resource plan by default.
  - Predefined maintenance windows use the `DEFAULT_MAINTENANCE_PLAN`

Oracle Scheduler is an enterprise job scheduler that helps simplify the scheduling of hundreds or even thousands of tasks. Oracle Scheduler (the Scheduler) is implemented by the procedures and functions in the `DBMS_SCHEDULER` PL/SQL package and can be invoked through Enterprise Manager Cloud Control. When a maintenance window opens, an Oracle Scheduler job is created for each maintenance task that is scheduled to run in that window.

Oracle Database Resource Manager (the Resource Manager) enables you to manage multiple workloads within a database that are contending for system and database resources. The elements of the Resource Manager include resource consumer groups, resource plans, and resource plan directives. The `DBMS_RESOURCE_MANAGER` PL/SQL package is used to create and maintain these elements.

By default, all predefined maintenance windows use the `DEFAULT_MAINTENANCE_PLAN` resource plan. Automated maintenance tasks run under its subplan named `ORA$AUTOTASK`. This subplan divides its portion of total resource allocation equally among the maintenance tasks. To change the resource allocation for automated maintenance tasks within a maintenance window, you must change the percentage of resources allocated to the `ORA$AUTOTASK` subplan in the resource plan for that window.

Predefined Automated Maintenance Tasks

Task	Description
Automatic Optimizer Statistics Collection	Collects statistics for all schema objects that have no statistics or only stale statistics
Optimizer Statistics Advisor	Analyzes how statistics are being gathered and suggests changes
Automatic Segment Advisor	Identifies segments that have reclaimable space and makes defragmenting recommendations
Automatic SQL Tuning Advisor	Examines the performance of high-load SQL statements and makes tuning recommendations
SQL Plan Management (SPM) Evolve Advisor	Evolves plans that have recently been added to the SQL plan baseline

Oracle Database includes the following predefined automated maintenance tasks:

- Automatic Optimizer Statistics Collection: Collects optimizer statistics for all schema objects in the database for which there are no statistics or only stale statistics. The statistics gathered by this task are used by the SQL query optimizer to improve the performance of SQL execution.
- Optimizer Statistics Advisor: Analyzes how statistics are being gathered and suggests changes that can be made to fine-tune statistics collection
- Automatic Segment Advisor: Identifies segments that have space available for reclamation, and makes recommendations on how to defragment those segments. You can also run the Segment Advisor manually to obtain more up-to-the-minute recommendations or recommendations on segments that the Automatic Segment Advisor did not examine for possible space reclamation.
- Automatic SQL Tuning Advisor: Examines the performance of high-load SQL statements and makes recommendations on how to tune those statements. You can configure this advisor to automatically implement SQL profile recommendations.
- SQL Plan Management (SPM) Evolve Advisor: Evolves plans that have recently been added to the SQL plan baseline. The advisor simplifies plan evolution by eliminating the requirement to do it manually.

All automated maintenance task job names begin with `ORA$AT`. For example, the job for the Automatic Segment Advisor might be called `ORA$AT_SA_SPC_SY_26`.

These tasks run in a set of predefined Oracle Scheduler job windows. The tasks are assigned to Oracle Database Resource Manager resource consumer groups. The Oracle Database Resource Manager resource plan that is in effect during the window controls the resources that the tasks are allowed to consume, such as CPU.

By default, all automated maintenance tasks are configured to run in all maintenance windows.

Maintenance Windows

- Maintenance windows are:
  - Contiguous time intervals during which automated maintenance tasks are run
  - Oracle Scheduler windows that belong to the window group named `MAINTENANCE_WINDOW_GROUP`

A maintenance window is a contiguous time interval during which automated maintenance tasks are run. Maintenance windows are Oracle Scheduler windows that belong to the window group named `MAINTENANCE_WINDOW_GROUP`.

Predefined Maintenance Windows

Task	Description
MONDAY_WINDOW	Starts at 10 PM on Monday and ends at 2 AM
TUESDAY_WINDOW	Starts at 10 PM on Tuesday and ends at 2 AM
WEDNESDAY_WINDOW	Starts at 10 PM on Wednesday and ends at 2 AM
THURSDAY_WINDOW	Starts at 10 PM on Thursday and ends at 2 AM
FRIDAY_WINDOW	Starts at 10 PM on Friday and ends at 2 AM
SATURDAY_WINDOW	Starts at 6 AM on Saturday and is 20 hours long
SUNDAY_WINDOW	Starts at 6 AM on Sunday and is 20 hours long

By default, there are seven predefined maintenance windows, each representing a day of the week. The `MAINTENANCE_WINDOW_GROUP` window group consists of these seven windows.

By default, the weekday maintenance window starts at 10 PM and lasts four hours. On Saturday and Sunday, the maintenance window starts at 6 AM and lasts for 20 hours. All attributes of the maintenance windows are customizable, including the start and end times, frequency, days of the week, and so on.

In the case of a very long maintenance window, all automated maintenance tasks, except Automatic SQL Tuning Advisor, are restarted every four hours. This feature ensures that maintenance tasks are run regularly, regardless of window size.

Automated Maintenance Tasks

Autotask maintenance process:

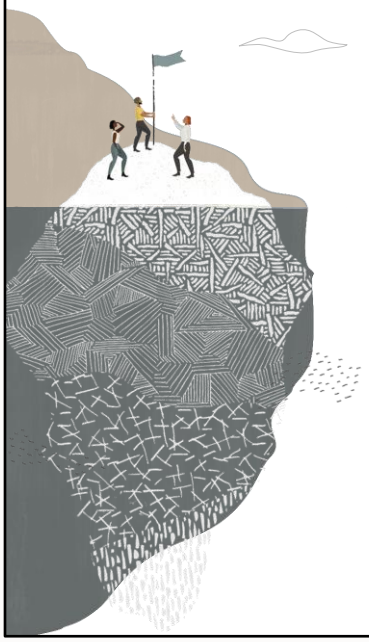
1. The maintenance window opens.
2. The Autotask background process schedules jobs.
3. Oracle Scheduler initiates jobs.
4. Oracle Resource Manager limits the impact of Autotask jobs.

By analyzing the information stored in the AWR, the database server can identify the need to perform routine maintenance tasks, such as optimizer statistics refresh. The automated maintenance tasks infrastructure enables the Oracle Database server to automatically perform such operations. It uses Oracle Scheduler to run such tasks in predefined maintenance windows.

When a maintenance window opens, Oracle Database creates an Oracle Scheduler job for each maintenance task that is scheduled to run in that window. Each job is assigned a job name that is generated at run time. When an automated maintenance task job finishes, it is deleted from the Oracle Scheduler job system. However, the job can still be found in the Scheduler job history.

The impact of automated maintenance tasks on normal database operations can be limited by associating a Database Resource Manager resource plan to the maintenance window.

Summary

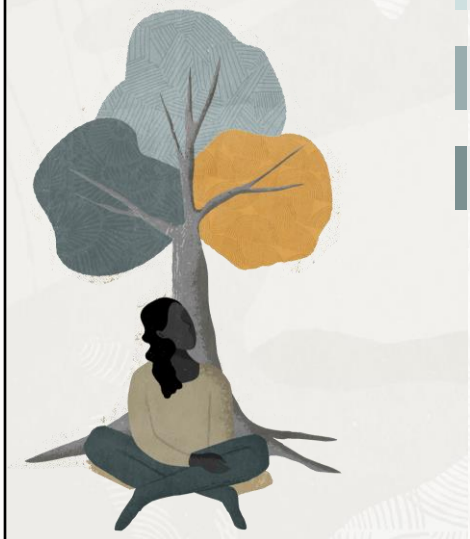


- Describe Oracle Database's proactive database maintenance infrastructure
- Discuss automated maintenance tasks
- Explain maintenance windows

ORACLE
University

Automated Maintenance Tasks: Managing Tasks and Windows

Objectives



- Enable and disable maintenance tasks
- Create, modify, and remove maintenance windows
- Reduce or increase resource allocation to automated maintenance tasks

Configuring Automated Maintenance Tasks

- You can perform the following configuration tasks:
  - Adjust the duration and start time of the maintenance window.
  - Control the resource plan that allocates resources to automated maintenance tasks during each window.
  - Enable or disable individual tasks in some or all maintenance windows.

The Automated Maintenance Tasks feature determines when—and in what order—tasks are performed. As a DBA, you can perform the following configuration tasks:

- Adjust the duration and start time of the maintenance window if the maintenance window turns out to be inadequate for the maintenance workload.
- Control the resource plan that allocates resources to automated maintenance tasks during each window.
- Enable or disable individual tasks in some or all maintenance windows.
- In a RAC environment, shift maintenance work to one or more instances by mapping maintenance work to a service. Enabling the service on a subset of instances shifts maintenance work to these instances.

Enterprise Manager is the preferred way to control automated maintenance tasks. However, you can also use the `DBMS_AUTO_TASK_ADMIN` package.

Enabling and Disabling Maintenance Tasks

- Enable or disable maintenance tasks for all maintenance windows:
  - Use the `ENABLE` and `DISABLE` procedures of the `DBMS_AUTO_TASK_ADMIN` package with the `WINDOW_NAME` argument set to `NULL`.
  - Use the `ENABLE` and `DISABLE` procedures with no arguments to enable or disable all automated maintenance tasks for all windows.
- Enable or disable maintenance tasks for specific maintenance windows:
  - Use the `ENABLE` and `DISABLE` procedures of the `DBMS_AUTO_TASK_ADMIN` package with the `WINDOW_NAME` argument set to a window name.

With a single operation, you can disable or enable a particular automated maintenance task for all maintenance windows. You can disable a particular automated maintenance task for all maintenance windows by calling the `DISABLE` procedure of the `DBMS_AUTO_TASK_ADMIN` PL/SQL package without supplying the `WINDOW_NAME` argument. To enable a maintenance task, use the `DBMS_AUTO_TASK_ADMIN.ENABLE` procedure. To enable or disable all automated maintenance tasks for all windows, call the `ENABLE` or `DISABLE` procedure with no arguments.

You can disable a maintenance task for a specific window by specifying the window name in the `WINDOW_NAME` argument.

Creating and Managing Maintenance Windows

- To create a new maintenance window:
  1. Use the DBMS_SCHEDULER.CREATE_WINDOW procedure to create the window.
  2. Use the DBMS_SCHEDULER.ADD_GROUP_MEMBER procedure to add the new window to the MAINTENANCE_WINDOW_GROUP window group.
- To change the attributes of a maintenance window:
  1. Use the DBMS_SCHEDULER.DISABLE procedure to disable the window.
  2. Use the DBMS_SCHEDULER.SET_ATTRIBUTE procedure to modify the window attributes.
  3. Use the DBMS_SCHEDULER.ENABLE procedure to re-enable the window.
- To remove a maintenance window, use the DBMS_SCHEDULER.REMOVE_GROUP_MEMBER procedure.

To create a new maintenance window, you must create an Oracle Scheduler window object and then add it to the MAINTENANCE_WINDOW_GROUP window group. Use the DBMS_SCHEDULER.CREATE_WINDOW package procedure to create the window and the DBMS_SCHEDULER.ADD_GROUP_MEMBER procedure to add the new window to the window group.

Use the DBMS_SCHEDULER.SET_ATTRIBUTE procedure to modify the attributes of a window. Note that you must use the DBMS_SCHEDULER.DISABLE subprogram to disable the window before making changes to it, and then re-enable the window with DBMS_SCHEDULER.ENABLE when you are finished. If you change a window when it is currently open, the change does not take effect until the next time the window opens.

You can use the DBMS_SCHEDULER.REMOVE_GROUP_MEMBER procedure to remove an existing maintenance window from the MAINTENANCE_WINDOW_GROUP window group. The window continues to exist but no longer runs automated maintenance tasks. Any other Oracle Scheduler jobs assigned to this window continue to run as usual.

Additional information about Oracle Scheduler is provided in the *Scheduling Tasks by Using Oracle Scheduler* learning module.

Resource Allocations for Automated Maintenance Tasks

- Automated maintenance tasks run under the ORA$AUTOTASK subplan of the DEFAULT_MAINTENANCE_PLAN resource plan.
- Any resource allocation that is unused by sessions in SYS_GROUP is shared by sessions belonging to OTHER_GROUPS and ORA$AUTOTASK in the percentages shown below:

Consumer Group/Subplan	Level 1	Maximum Utilization Limit
ORA$AUTOTASK	5%	90
OTHER_GROUPS	20%	-
SYS_GROUP	75%	-

- ORA$AUTOTASK cannot be allocated more than 90% of the CPU resources.

When a maintenance window opens, DEFAULT_MAINTENANCE_PLAN in the Resource Manager is automatically set to control the amount of CPU resources used by automated maintenance tasks. To be able to give different priorities to each possible task during a maintenance window, various consumer groups are assigned to DEFAULT_MAINTENANCE_PLAN.

By default, all predefined maintenance windows use the DEFAULT_MAINTENANCE_PLAN resource plan. Automated maintenance tasks run under its subplan called ORA$AUTOTASK. This subplan divides its portion of total resource allocation equally among the maintenance tasks.

In this plan, any sessions in the SYS_GROUP consumer group get priority. Any resource allocation that is unused by sessions in SYS_GROUP is then shared by sessions belonging to the other consumer groups and subplans in the plan. Of that allocation, 5% goes to maintenance tasks and 20% goes to user sessions. The maximum utilization limit for ORA$AUTOTASK is 90. Therefore, even if the CPU is idle, this group/plan cannot be allocated more than 90% of the CPU resources.

Additional information about the Resource Manager is provided in the *Managing Database Resources* learning module.

Changing Resource Allocations for Maintenance Tasks

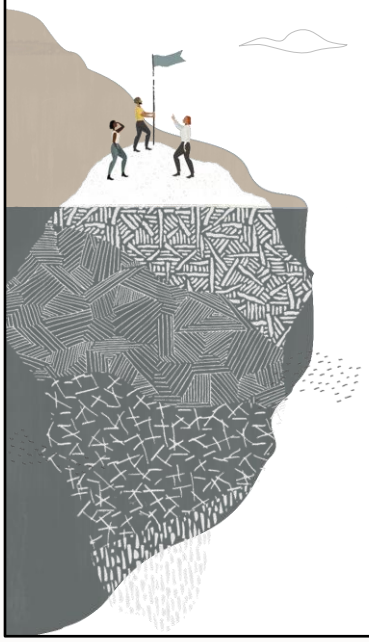
- Change the percentage of resources allocated to the ORA$AUTOTASK subplan in the resource plan for the window of interest.
- Adjust the resource allocation for one or more subplans or consumer groups in the window's resource plan so that the resource allocation at the top level of the plan adds up to 100%.

To change the resource allocation for automated maintenance tasks within a maintenance window, you must change the percentage of resources allocated to the ORA$AUTOTASK subplan in the resource plan for that window. You must also adjust the resource allocation for one or more subplans or consumer groups in the window's resource plan so that the resource allocation at the top level of the plan adds up to 100%.

By default, the resource plan for each predefined maintenance window is DEFAULT_MAINTENANCE_PLAN. However, you can assign any resource plan to any maintenance window. If you do change a maintenance window resource plan, you must include the ORA$AUTOTASK subplan in the new plan.

See the *Oracle Database Administrator's Guide* for details on modifying resource plans and subplans.

Summary



- Enable and disable maintenance tasks
- Create, modify, and remove maintenance windows
- Reduce or increase resource allocation to automated maintenance tasks

Practice Overview

- Enabling and Disabling Automated Maintenance Tasks
- Modifying the Duration of a Maintenance Window

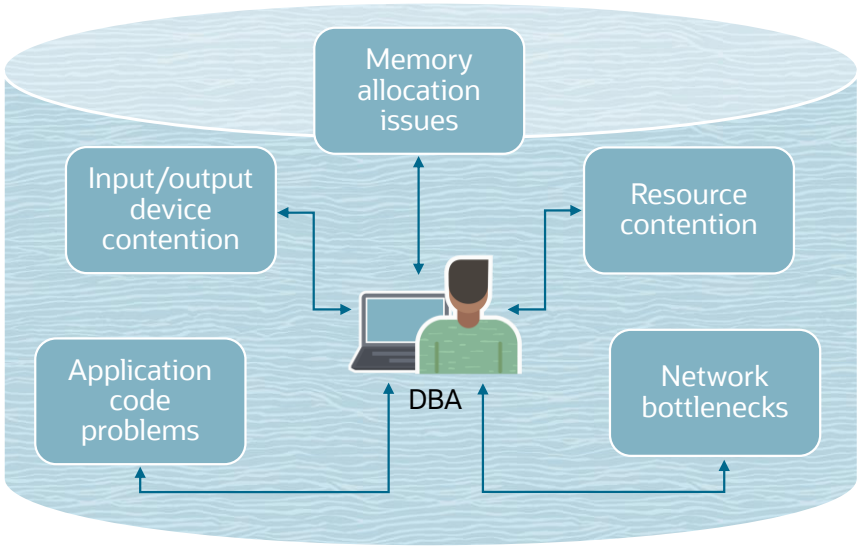
ORACLE
University

Database Monitoring and Tuning Performance Overview

Objectives

- Describe the activities that you perform to manage database performance
- Explain the Oracle performance tuning methodology

Performance Management Activities

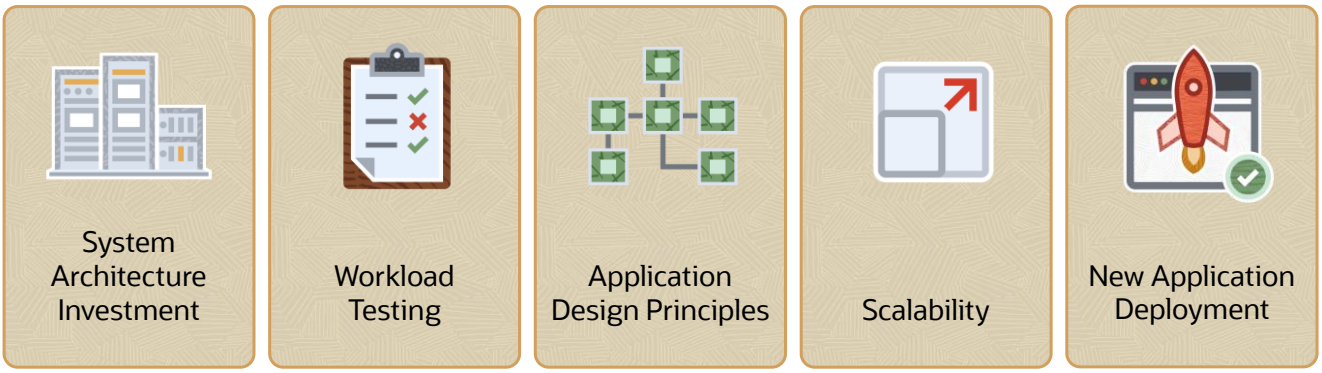


Performance management includes several activities.

- Performance planning is the process of establishing the environment: the hardware, software, operating system, network infrastructure, and so on.
- Performance monitoring is an activity that helps the DBA locate bottlenecks and correct problem areas.
- Instance tuning is the actual adjustment of Oracle Database parameters and operating system (OS) parameters to gain better performance of the Oracle Database.
- SQL tuning involves making your application submit efficient SQL statements. SQL tuning is performed for the application as a whole, as well as for individual statements. At the application level, you want to be sure that different parts of the application are taking advantage of each other's work and not competing for resources unnecessarily.

A DBA can look at hundreds of performance measurements, covering everything from network performance and disk input/output (I/O) speed to the time spent working on individual application operations. These performance measurements are commonly referred to as database metrics.

Performance Planning Considerations



There are many facets to performance planning. Planning must include a balance between performance (speed), cost, and reliability.

Investment in System Architecture: You must consider the investment in your system architecture—the hardware and software infrastructure needed to meet your requirements. This, of course, requires analysis to determine the value of your given environment, application, and performance requirements. For example, the number of hard drives and controllers has an impact on the speed of data access.

Scalability: The ability of an application to scale is also important. This means that you are able to handle more and more users, clients, sessions, or transactions without incurring a huge impact on overall system performance. The most obvious violator of scalability is serializing operations among users. If all users go through a single path one at a time, then, as more users are added, there are definitely adverse effects on performance. This is because more and more users line up to go through that path. Poorly written SQL also affects scalability. It requires many users to wait for inefficient SQL to complete, each user competing with the other on a large number of resources that they are not actually in need of.

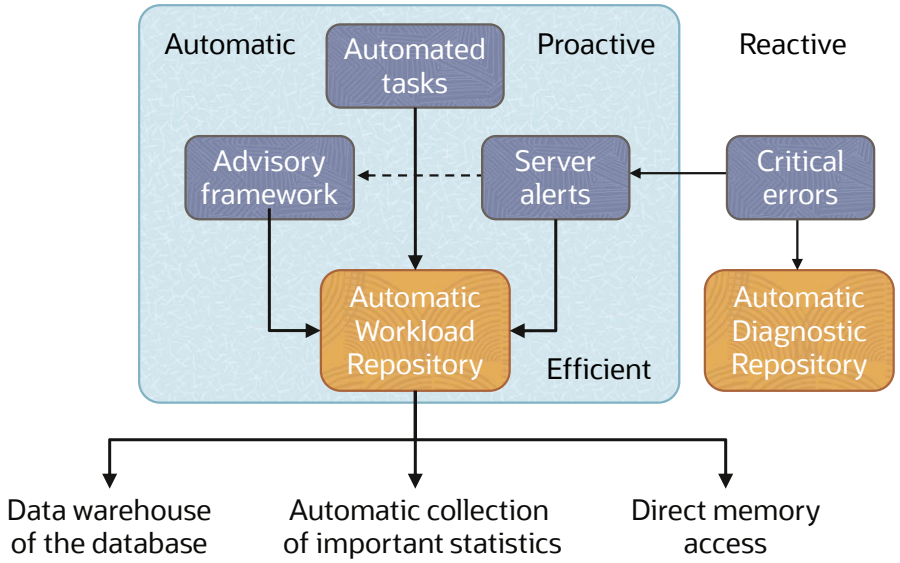
Application Design Principles: The principles of application design can greatly affect performance. Simplicity of design, use of views and indexes, and data modeling are all very important.

Workload Testing: Any application must be tested under a representative production workload. This requires estimating database size and workload and generating test data and system load.

Deployment of New Applications: Performance must be considered as new applications (or new versions of applications) are deployed. Sometimes, design decisions are made to maintain compatibility with old systems during the rollout. A new database should be configured (on the basis of the production environment) specifically for the applications that it hosts.

A difficult and necessary task is testing the existing applications when changing the infrastructure (for example, upgrading the database to a newer version or changing the operating system or server hardware). Before the application is deployed for production in the new configuration, you want to know the impact. The application will almost certainly require additional tuning. You need to know that the critical functionality will perform, without errors.

Database Maintenance



Proactive database maintenance is made easy by the sophisticated infrastructure of the Oracle Database, including the following main elements:

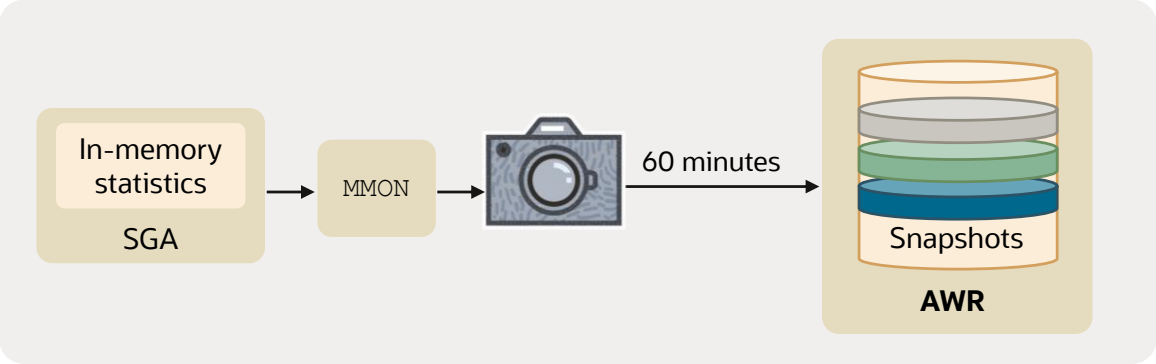
- The Automatic Workload Repository (AWR) is a built-in repository in each Oracle database. At regular intervals, the Oracle Database server takes a snapshot of all its vital statistics and workload information and stores this data in the AWR. The captured data can be analyzed by you, by the database server itself, or by both.
- Using automated tasks, the database server performs routine maintenance operations, such as regular backups, refreshing optimizer statistics, and database health checks.

Reactive database maintenance includes critical errors and conditions discovered by database health checkers:

- For problems that cannot be resolved automatically and require administrators to be notified (such as running out of space), the Oracle Database server provides server-generated alerts. The Oracle Database server, by default, monitors itself and sends out alerts to notify you of problems. The alerts notify you and often also provide recommendations on how to resolve the reported problem. The DBA can also be alerted by users whose transactions are locked by other users' transactions and are waiting for locks to be released.
- Recommendations are generated from several advisors, each of whom is responsible for a subsystem. For example, there are memory, segment, and SQL advisors.

Automatic Workload Repository (AWR)

- Built-in repository of performance information
- Snapshots of database metrics taken every 60 minutes and retained for eight days
- Foundation of all self-management functions



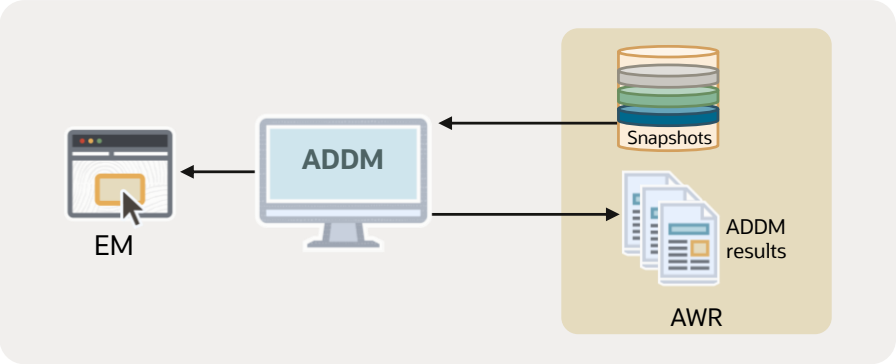
The Automatic Workload Repository (AWR) is the infrastructure that provides services to Oracle Database components to collect, maintain, and use statistics for problem detection and self-tuning purposes. You can view it as a data warehouse for database statistics, metrics, and so on.

Every 60 minutes (by default), the database automatically captures statistical information from the SGA and stores it in the AWR in the form of snapshots. These snapshots are stored on disk by a background process called Manageability Monitor (MMON). By default, snapshots are retained for eight days. You can modify both the snapshot interval and the retention interval.

The AWR contains hundreds of tables, all belonging to the `SYS` schema and stored in the `SYSAUX` tablespace. Oracle recommends that the repository be accessed only through Enterprise Manager or the `DBMS_WORKLOAD_REPOSITORY` package to work with the AWR. Direct data manipulation language (DML) commands against the repository tables are not supported.

Automatic Database Diagnostic Monitor (ADDM)

- Runs after each AWR snapshot
- Monitors the instance, detects bottlenecks
- Stores results in the AWR



Unlike the other advisors, the ADDM runs automatically after each AWR snapshot. Each time a snapshot is taken, the ADDM performs an analysis of the period corresponding to the last two snapshots. The ADDM proactively monitors the instance and detects most bottlenecks before they become a significant problem.

Note: For the information to be valid, the instance should not have been shut down between the two snapshots.

In many cases, the ADDM recommends solutions for detected problems and even quantifies the benefits for the recommendations.

Some common problems that are detected by the ADDM are:

- CPU bottlenecks
- Poor Oracle Net connection management
- Lock contention
- Input/output (I/O) capacity
- Undersized database instance memory structures
- High-load SQL statements

The results of each ADDM analysis are stored in the AWR and are also accessible through Enterprise Manager.

Configuring Automatic ADDM Analysis at the PDB Level

1. Enable PDB AWR snapshot creation on the CDB root and on each PDB:

```
SQL> ALTER SYSTEM SET awr_pdb_autoflush_enabled = TRUE;
```

2. Set the AWR snapshot interval to greater than 0 at the PDB level:

```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> EXEC dbms_workload_repository.modify_snapshot_settings(interval => 60)
```

3. Execute the ADDM task (manually when required):

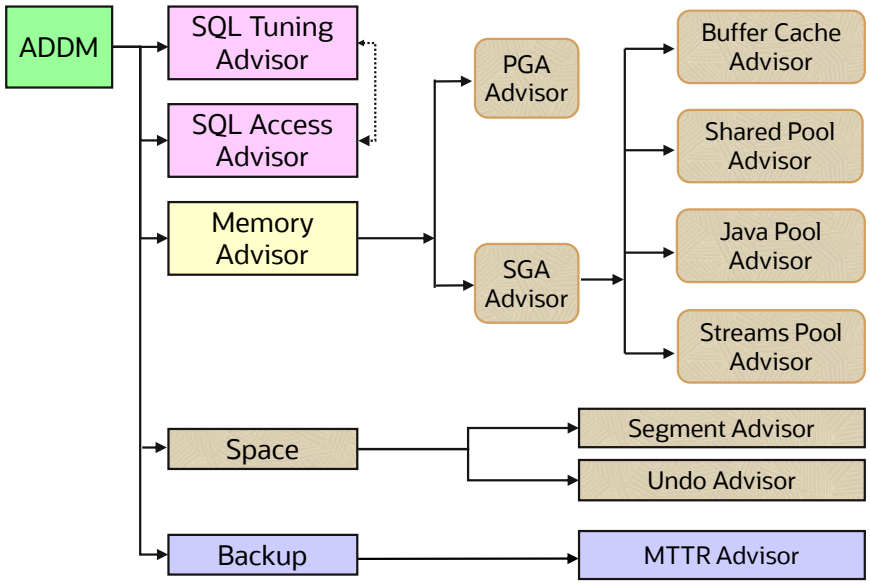
```
SQL> CONNECT sys@PDB1 AS SYSDBA
SQL> EXEC DBMS_ADDM.ANALYZE_DB(:tname, begin_snapshot =>1, end_snapshot =>2)
```

ADDM can run automatically on a PDB after AWR snapshots are enabled in the PDB. AWR snapshots are not enabled by default on a PDB.

When AWR snapshots are enabled in a PDB and an ADDM analysis runs in the PDB, some parts of the ADDM report and recommendations differ from the result of an ADDM execution in the CDB root. A PDB analysis means that the ADDM task is analyzing the AWR data of a PDB, and only PDB-specific findings and recommendations are produced. Enabling AWR snapshots on a PDB does not change the ADDM report on a CDB root.

AWR snapshots on a PDB are stored in the PDB, in whichever tablespace is defined. No one can view PDB snapshots from the CDB root for security purposes.

Advisory Framework



Advisors provide you with useful feedback about resource utilization and performance for their respective server components. For example, Memory Advisor provides a recommended value for the `MEMORY_TARGET` initialization parameter, which controls the total amount of memory used by the Oracle database instance.

By building on the data captured in the AWR, the ADDM enables the Oracle Database server to diagnose its own performance and determine how identified problems can be resolved. ADDM runs automatically after each AWR statistics capture. It can potentially call other advisors.

Here are the major benefits that are provided by the advisor infrastructure:

- All advisors use a uniform interface.
- All advisors have a common data source and results storage by using the workload repository.

Not all advisors are shown in the slide (for example, Data Recovery Advisor and SQL Repair Advisor are not listed).

Automatic Database Diagnostic Monitor (ADDM)

The ADDM is a server-based expert that reviews database performance every 60 minutes. Its goal is to detect possible system bottlenecks early and recommend fixes before system performance degrades noticeably.

Memory Advisor

Memory Advisor is a collection of several advisory functions that help determine the best settings for the total memory used by the database instance. The System Global Area (SGA) has a set of advisors for the shared pool, database buffer cache, Java pool, and streams pool. The Java pool and streams pool advisors are not exposed on the Enterprise Manager Memory Advisor page. There is an advisor for the Program Global Area (PGA). In addition to the advisory functions, this advisor provides a central point of control for the large pool and the Java pool.

Mean-Time-To-Recover (MTTR) Advisor

Using MTTR Advisor, you set the length of time required for the database to recover after an instance crash.

Segment Advisor

This advisor looks for tables and indexes that consume more space than they require. The advisor checks for inefficient space consumption at the tablespace or schema level and produces scripts to reduce space consumption where possible.

SQL Access Advisor

This advisor analyzes all SQL statements that are issued in a given period and suggests the creation of additional indexes, materialized views, or partitioning that will improve performance.

SQL Tuning Advisor

This advisor analyzes an individual SQL statement and makes recommendations for improving its performance. Recommendations may include actions, such as rewriting the statement, changing the instance configuration, or adding indexes.

Undo Management Advisor

With Undo Management Advisor, you can determine the undo tablespace size that is required to support a given retention period. Undo management and the use of the advisor is covered in the lesson titled "Managing Undo Data."

Data Recovery Advisor

This advisor automatically diagnoses persistent data failures, presents repair options to the user, and executes repairs at the user's request. The purpose of Data Recovery Advisor is to reduce the mean time to recover (MTTR) and provide a centralized tool for automated data repair.

SQL Repair Advisor

You run SQL Repair Advisor after a SQL statement fails with a critical error that generates a problem in the Automatic Diagnostic Repository. The advisor analyzes the statement and, in many cases, recommends a patch to repair the statement. If you implement the recommendation, the applied SQL patch circumvents the failure by causing the query optimizer to choose an alternative execution plan for future executions. This is done without changing the SQL statement itself.

Performance Tuning Methodology

System Health and OS Statistics

Top Down Approach: Design Application Database Instance

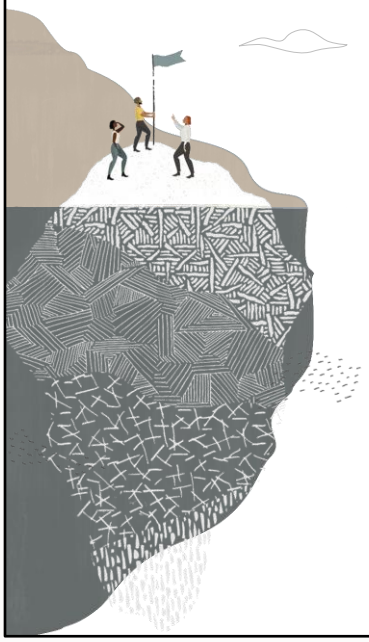
Tune Areas with Greatest Benefit

Oracle has developed a tuning methodology based on years of experience. The basic steps are:

1. Check the OS statistics and general machine health before tuning the instance to be sure that the problem is in the database.
2. Tune from the top down. Start with the design, then the application, and then the instance. For example, try to eliminate full table scans that cause I/O contention before tuning the tablespace layout on disk. This activity often requires access to the application code.
3. Tune the area with the greatest potential benefit. The tuning methodology presented in this course is simple. Identify the biggest bottleneck and tune it. Repeat this step. All the various tuning tools have some way to identify the SQL statements, resource contention, or services that are taking the most time. The Oracle database provides a time model and metrics to automate the process of identifying bottlenecks. The advisors available in Oracle Database use this methodology.
4. Stop tuning when you meet your goal. This step implies that you set tuning goals.

This is a general approach to tuning the database instance and may require multiple passes.

Summary



- Describe the activities that you perform to manage database performance
- Explain the Oracle performance tuning methodology

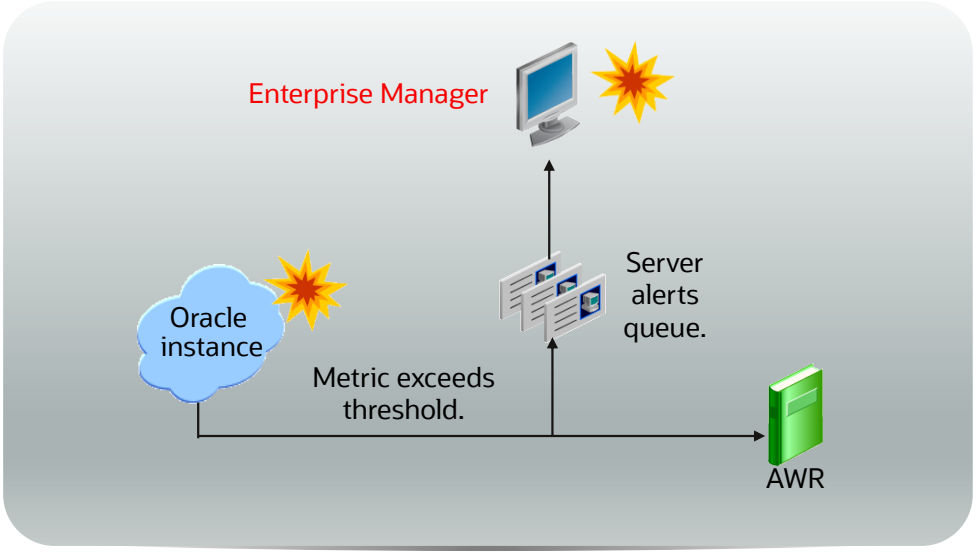
ORACLE
University

Monitoring Database Performance

Objectives

- Use performance views and tools to monitor database instance performance
- Describe the server statistics and metrics that are collected by the Oracle Database server

Server-Generated Alerts



Alerts are notifications of when a database is in an undesirable state and needs your attention. By default, the Oracle Database server provides alerts via Enterprise Manager. Optionally, Enterprise Manager can be configured to send an email message to the administrator about problem conditions as well as display alert information on the console.

You can also set thresholds on many of the pertinent metrics for your system. Oracle Database proactively notifies you if the database deviates sufficiently from normal readings to reach those thresholds. An early notification of potential problems enables you to respond quickly and, in many cases, resolve issues before users even notice them.

Approximately 60 metrics are monitored by default, among which are:

- Broken Job Count
- Database Time Spent Waiting (%)
- Dump Area Used (%)
- SQL Response Time (%) Compared to Baseline
- Tablespace Used (%)
- Generic Incident

A few additional key metrics can provide early problem notification:

- Average File Read Time (centiseconds)
- Response Time (per transaction)
- Wait Time (%)

Setting Metric Thresholds

- View and change threshold settings for the server alert metrics by using:
  - The `GET_THRESHOLD` and `SET_THRESHOLD` procedures of the `DBMS_SERVER_ALERT` PL/SQL package
  - The Metric and Collection Settings page in Enterprise Manager Cloud Control

You can set a desired warning and critical threshold values for a number of predefined metrics. The appropriate alerts appear when the database reaches your specified values.

You can use the `DBMS_SERVER_ALERT` package, which enables you to view and change threshold settings. Use the `GET_THRESHOLD` procedure to see the current threshold settings for a specified metric. Use the `SET_THRESHOLD` procedure to modify the threshold values. Refer to *Oracle Database PL/SQL Packages and Types Reference* for a complete list of supported metrics and parameters for each procedure.

You can also use the Metric and Collection Settings page in Enterprise Manager Cloud Control to view and set thresholds. The thresholds that are already set appear in the “Metrics with thresholds” list. By default, approximately 60 metrics have preset thresholds; you may change these as needed. The “All metrics” list shows the metrics that do not have thresholds set.

You can also change the scheduled collection interval for a specific collection schedule. Be aware that each schedule affects a group of metrics.

Reacting to Alerts

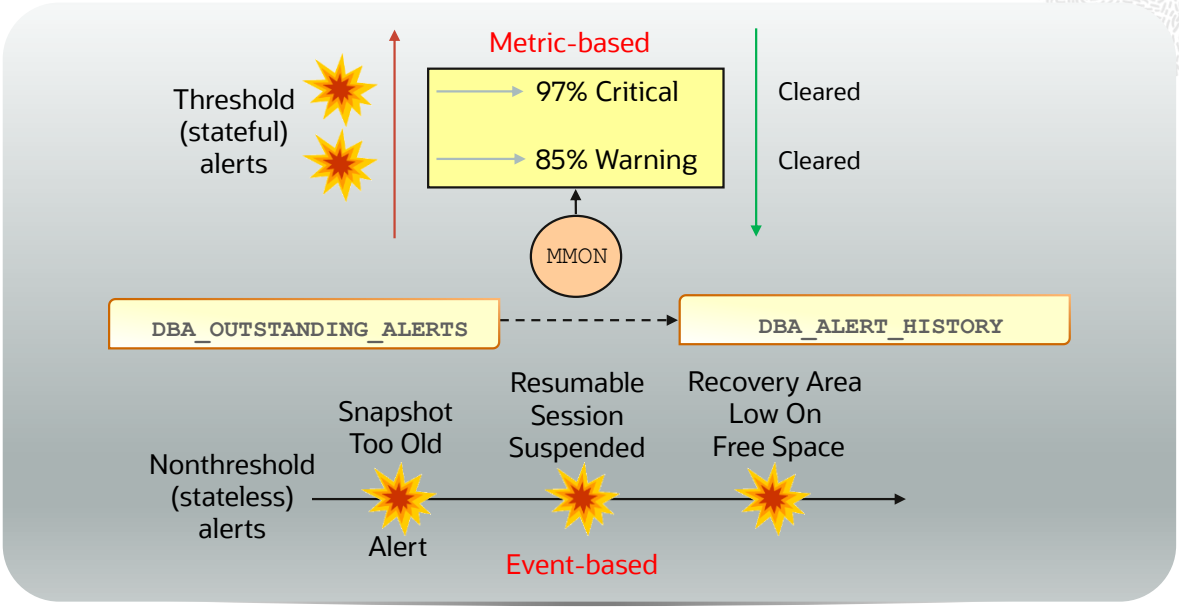
- If necessary, you should gather more input (for example, by running ADDM or another advisor).
- Investigate critical errors.
- Take corrective measures.
- Acknowledge alerts that are not automatically cleared.

When you receive an alert, follow the recommendations that it provides. You can also consider running the ADDM (or another advisor as appropriate) to obtain more detailed diagnostics of system or object behavior.

Alerts and incidents are generated for critical errors. Critical errors usually generate incidents that are collected into problems.

Most alerts (such as “Out of Space”) are cleared automatically when the cause of the problem disappears. However, other alerts (such as Generic Alert Log Error) are sent to you for notification and must be acknowledged by you. After taking the necessary corrective measures, you acknowledge an alert by clearing or purging it. Clearing an alert sends the alert to the Alert History, which is accessible from the Monitoring submenu. Purging an alert removes it from the Alert History.

Alert Types and Clearing Alerts



There are two kinds of server-generated alerts: threshold and nonthreshold.

Most server-generated alerts are configured by setting a warning and critical threshold values on database metrics. You can define thresholds for more than 120 metrics, including the following:

- Physical Reads Per Sec
- User Commits Per Sec
- SQL Service Response Time

Except for the Tablespace Space Usage metric, which is database-related, the other metrics are instance-related. Threshold alerts are also referred to as **stateful alerts,** which are automatically cleared when an alert condition clears. Stateful alerts appear in `DBA_OUTSTANDING_ALERTS` and, when cleared, go to `DBA_ALERT_HISTORY`.

Other server-generated alerts correspond to specific database events, such as `ORA-*` errors, "Snapshot too old" errors, Recovery Area Low On Free Space, and Resumable Session Suspended. These are non-threshold-based alerts, also referred to as **stateless alerts**. Stateless alerts go directly to the History table.

Database Server Statistics and Metrics

Cumulative statistics:

- Wait events with time information
- Time model

Sampled statistics:

- Active session history
- Statistics by session, SQL, and service
- Other dimensions

Metrics: Statistic rates

The Oracle Database server software captures information about its own operation. Three major types of data are collected: cumulative statistics, metrics, and sampled statistics.

Cumulative statistics are counts and timing information of a variety of events that occur in the database server. Some are quite important, such as buffer busy waits. Others have little impact on tuning, such as index block split. The most important events for tuning are usually the ones showing the greatest cumulative time values. The statistics in Oracle Database are correlated by the use of a time model. The time model statistics are based on a percentage of DB time, giving them a common basis for comparison.

Metrics are statistic counts per unit. The unit could be time (such as seconds), transaction, or session. Metrics provide a base to proactively monitor performance. You can set thresholds on a metric, causing an alert to be generated. For example, you can set thresholds for when the reads per millisecond exceed a previously recorded peak value or when the archive log area is 95% full.

Sampled statistics are gathered automatically when `STATISTICS_LEVEL` is set to `TYPICAL` or `ALL`. Sampled statistics allow you to look back in time. You can view session and system statistics that were gathered in the past, in various dimensions, even if you had not thought of specifying data collection for these beforehand.

Performance Monitoring

- Enterprise Manager Database Express
- Enterprise Manager Cloud Control
- Performance views

Instance/Database	Disk
V$DATABASE	V$DATAFILE
V$INSTANCE	V$FILESTAT
V$PARAMETER	V$LOG
V$SPPARAMETER	V$LOG_HISTORY
V$SYSTEM_PARAMETER	V$DBFILE
V$PROCESS	V$TEMPFILE
V$BGPROCESS	V$TEMPSEG_USAGE
V$PX_PROCESS_SYSSTAT	V$SEGMENT_STATISTICS
V$SYSTEM_EVENT	

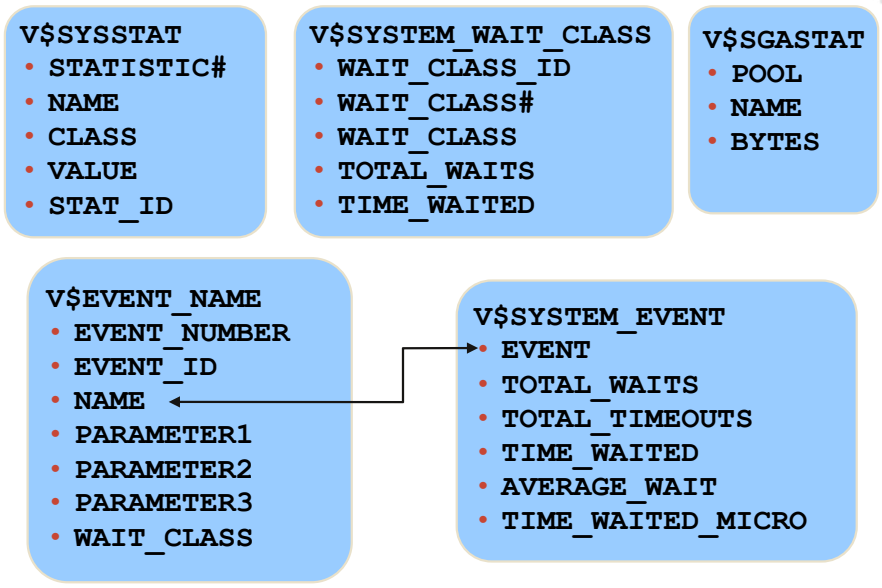
Memory	Contention
V$BUFFER_POOL_STATISTICS	V$LOCK
V$LIBRARYCACHE	V$UNDOSTAT
V$SGAINFO	V$WAITSTAT
V$PGASTAT	V$LATCH

You can respond to changes in performance only if you know the performance has changed. Oracle Database provides several ways to monitor the current performance of the database instance.

- **Enterprise Manager Database Express:** The database home page provides a quick check of the health of the instance and the server, with graphs showing CPU usage, active sessions, memory, and data storage usage. The home page also shows any alerts that have been triggered. Additional details are available on the Performance Hub page. As mentioned earlier in the lesson, ADDM analysis results are accessible through Enterprise Manager.
- **Enterprise Manager Cloud Control:** Enterprise Manager Cloud Control also provides performance monitoring capabilities.
- **Performance views:** You can access these views directly with SQL*Plus. Occasionally, you may need to access these views for some details about the raw statistics.

See *Oracle Database Performance Tuning Guide* and *Oracle Database Reference* for details and examples.

Viewing Statistics Information



To effectively diagnose performance problems, statistics must be available. The Oracle Database server generates many types of cumulative statistics for the system, sessions, and individual SQL statements at the instance level. The Oracle Database server also tracks cumulative statistics on segments and services. When analyzing a performance problem in any of these scopes, you typically look at the change in statistics (delta value) over the period of time you are interested in.

Note: Instance statistics are dynamic and are reset at every instance startup. These statistics can be captured at a point in time and held in the database in the form of snapshots.

Wait Event Statistics

All possible wait events are cataloged in the `V$EVENT_NAME` view.

Cumulative statistics for all sessions are stored in `V$SYSTEM_EVENT`, which shows the total waits for a particular event since instance startup.

When you are troubleshooting, you need to know whether a process has waited for any resource.

Systemwide Statistics

All systemwide statistics are cataloged in the `V$STATNAME` view. Over 400 statistics are available in Oracle Database.

The server displays all calculated system statistics in the `V$SYSSTAT` view. You can query this view to find cumulative totals since the instance started.

Systemwide statistics are classified by the tuning topic and the debugging purpose. The classes include general instance activity, redo log buffer activity, locking, database buffer cache activity, and so on. Each of the system statistics can belong to more than one class, so you cannot do a simple join on `V$SYSSTATS.CLASS and V$SYSTEM_WAIT_CLASS.WAIT_CLASS#`.

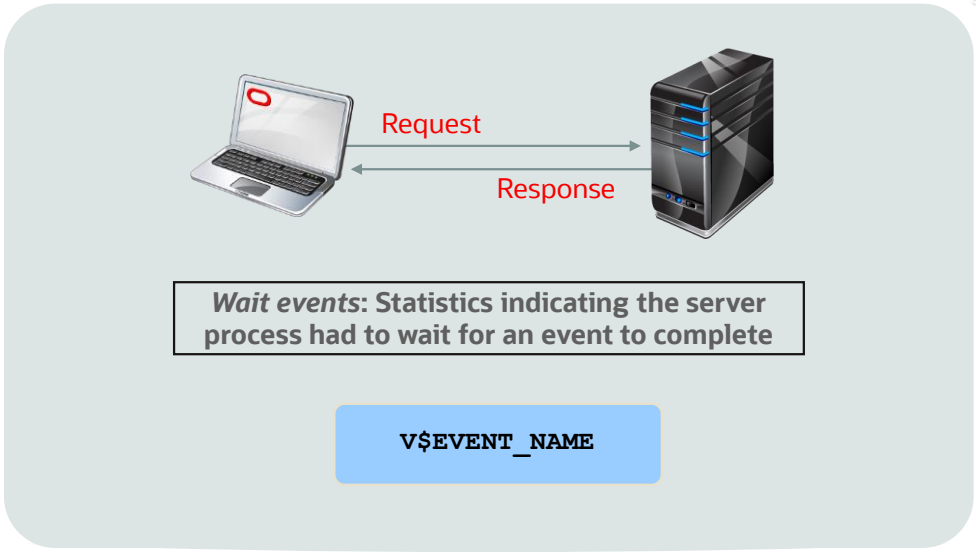
You can also view all wait events for a particular wait class by querying `V$SYSTEM_WAIT_CLASS`.

SGA Global Statistics

The server displays all calculated memory statistics in the V$SGASTAT view. You can query this view to find cumulative totals of detailed SGA usage since the instance started.

When the STATISTICS_LEVEL parameter is set to BASIC, the value of the TIMED_STATISTICS parameter defaults to FALSE. Timing information is not collected for wait events, and much of the performance-monitoring capability of the database is disabled. The explicit setting of TIMED_STATISTICS overrides the value derived from STATISTICS_LEVEL.

Monitoring Wait Events



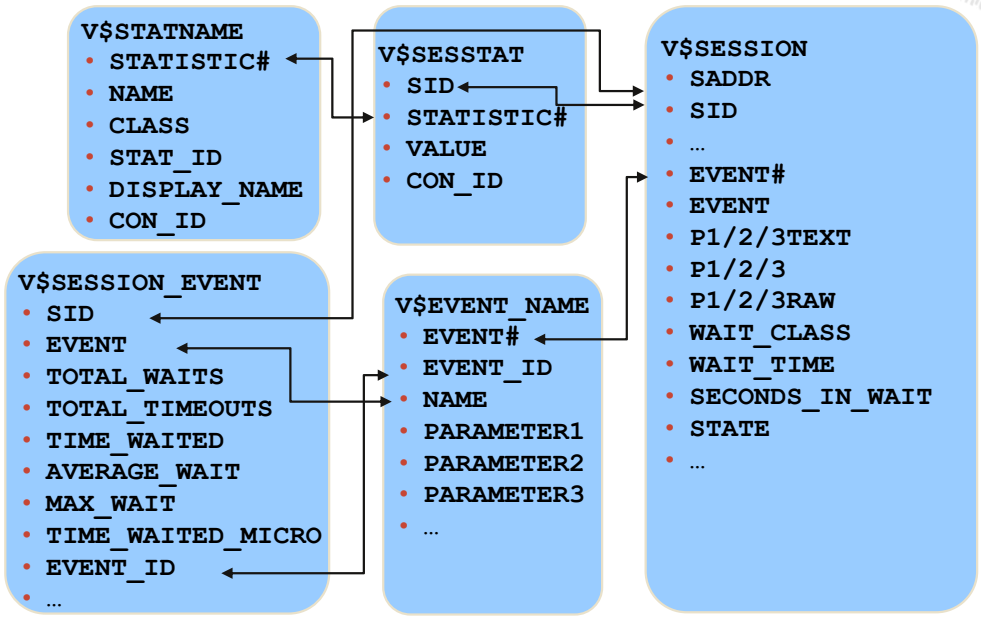
Wait events are statistics that are incremented by a server process or thread to indicate that it had to wait for an event to complete before being able to continue processing. Wait event data reveals various symptoms of problems that might be impacting performance, such as latch contention, buffer contention, and I/O contention. Remember that these are only symptoms of problems, not the actual causes.

A collection of wait events provides information about the sessions or processes that had to wait or must wait for different reasons.

Wait events are grouped into classes. The wait event classes include Administrative, Application, Cluster, Commit, Concurrency, Configuration, Idle, Network, Other, Scheduler, System I/O, and User I/O.

Wait events are listed in the `V$EVENT_NAME` view. There are more than 800 wait events in the Oracle Database, including free buffer wait, latch free, buffer busy waits, DB file sequential read, and DB file scattered read.

Monitoring Sessions



You can display current session information for each user logged on by querying `V$SESSION`. For example, you can use `V$SESSION` to determine whether a session represents a user session or was created by a database server process (background).

You can query either `V$SESSION` or `V$SESSION_WAIT` to determine the resources or events for which active sessions are waiting.

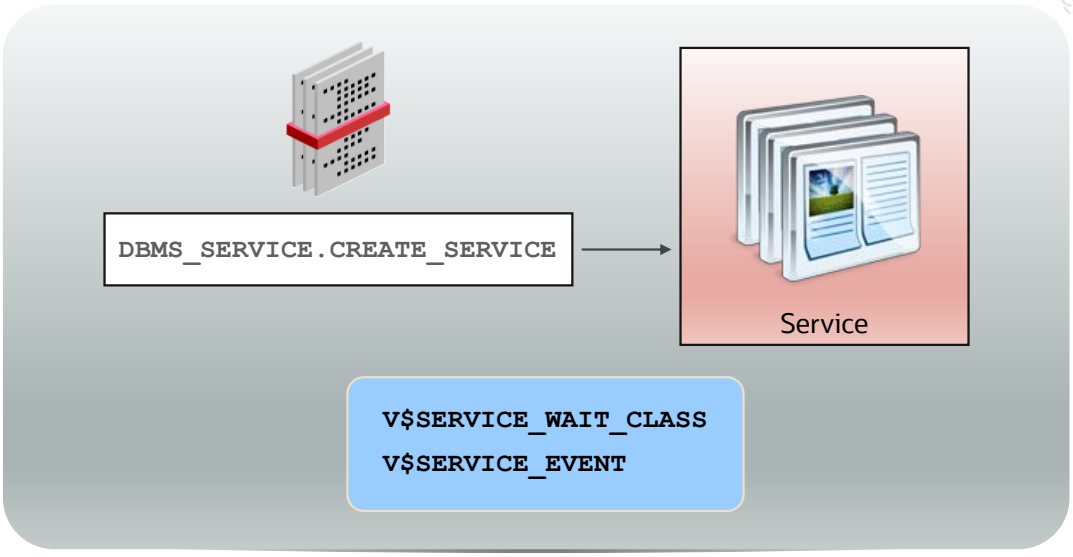
You can view user session statistics in `V$SESSTAT`. The `V$SESSION_EVENT` view lists information about waits for an event by session.

Cumulative values for instance statistics are generally available through dynamic performance views, such as `V$SESSTAT` and `V$SYSSTAT`. Note that the cumulative values in dynamic views are reset when the database instance is shut down.

The `V$MYSTAT` view displays the statistics of the current session.

You can also query `V$SESSMETRIC` to display performance metric values for all active sessions. This view lists performance metrics, such as CPU usage, number of physical reads, number of hard parses, and the logical read ratio.

Monitoring Services



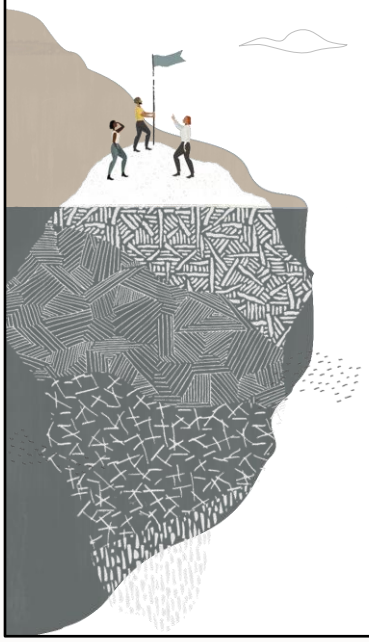
In a multi-tier environment where there is an application server that is pooling database connections, viewing sessions may not provide the information you need to analyze performance. Grouping sessions into service names enables you to monitor performance more accurately. Regardless of the session that was used for a particular request, if it connected via one of these services, the performance data of the session is captured under that service name.

You can define a service in the database by using the `DBMS_SERVICE` package and can use the net service name to assign applications to a service.

Two views provide the same information that their like-named session counterparts provide, except that the information is presented at the service level rather than at the session level.

- `V$SERVICE_WAIT_CLASS` shows wait statistics for each service, broken down by wait class.
- `V$SERVICE_EVENT` shows the same information as `V$SERVICE_WAIT_CLASS`, except that it is further broken down by event ID.

Summary



- Use performance views and tools to monitor database instance performance
- Describe the server statistics and metrics that are collected by the Oracle Database server

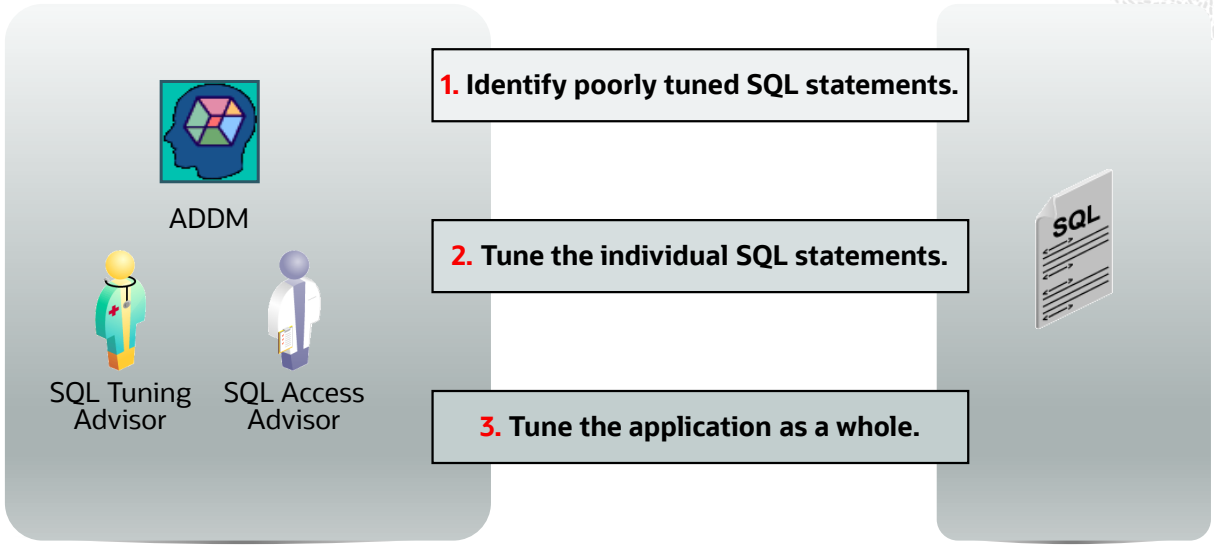
ORACLE
University

Analyzing SQL and Optimizing Access Paths

Objectives

- Describe the SQL tuning methodology
- Manage optimizer statistics
- Use SQL Tuning Advisor to identify and tune SQL statements that are using the most resources
- Use SQL Access Advisor to tune a workload

SQL Tuning Process

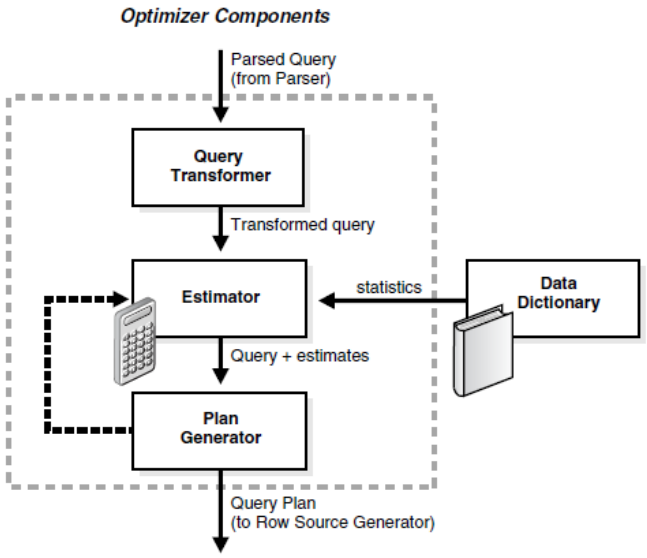


SQL Tuning Process

1. **Identify poorly tuned SQL statements:** Generally, the tuning effort that yields the most benefit is SQL tuning. Poorly tuned SQL uses more resources than required. This inefficiency prevents scalability, uses more OS and database resources, and increases response time. To tune poorly tuned SQL statements, they must be identified and then tuned. SQL statements can be tuned individually, but often the solution that optimizes one statement can hurt the performance of several others. The SQL statements that use the most resources are, by definition, the statements in need of tuning. These are statements that have the longest elapsed time, use the most CPU, or do the most physical or logical reads. Automatic Database Diagnostic Monitor (ADDM) can detect high-load SQL statements.
2. **Tune the individual statements:** Tune the individual statements by checking the optimizer statistics; check the explain plan for the most efficient access path; test alternative SQL constructions; and test possible new indexes, materialized views, and partitioning. SQL Tuning Advisor and SQL Access Advisor, described later in this lesson, can help with this task.
3. **Tune the application as a whole:** Test the application as a whole by using the tuned SQL statements. Is the overall performance better?

The methodology is sound, but tedious. Tuning an individual statement is not difficult. Testing the overall impact of the individual statement tuning on an application can be very difficult.

Oracle Optimizer



The optimizer is the part of the Oracle Database server that creates the execution plan for a SQL statement. The determination of the execution plan is an important step in the processing of any SQL statement and can greatly affect execution time.

The execution plan is a series of operations that are performed in sequence to execute the statement. The optimizer considers many factors related to the referenced objects and the conditions specified in the query. The information necessary for the optimizer includes:

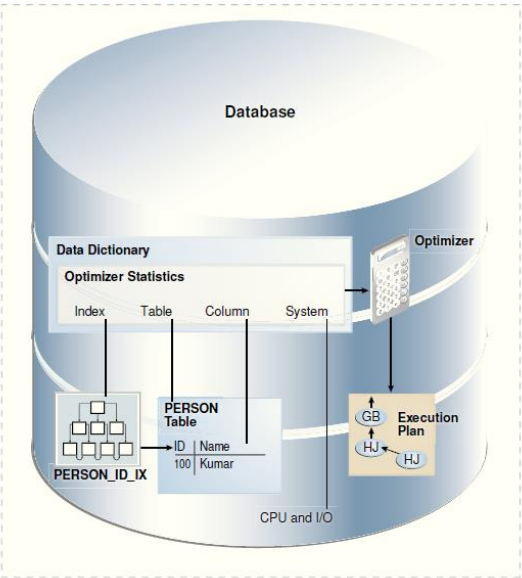
- Statistics gathered for the system (I/O, CPU, and so on) as well as schema objects (number of rows, index, and so on)
- Information in the dictionary
- `WHERE` clause qualifiers
- Hints supplied by the developer

The optimizer:

- Evaluates expressions and conditions
- Uses object and system statistics
- Decides how to access the data and join tables
- Determines the most efficient path

When you use diagnostic tools, such as Enterprise Manager, `EXPLAIN PLAN`, and SQL*Plus `AUTOTRACE`, you can see the execution plan that the optimizer chooses.

Optimizer Statistics



Optimizer statistics include table, column, index, and system statistics. Statistics for tables and indexes are stored in the data dictionary. These statistics are not intended to provide real-time data. They provide the optimizer a statistically correct snapshot of data storage and distribution, which the optimizer uses to make decisions on how to access data.

The statistics that are collected include:

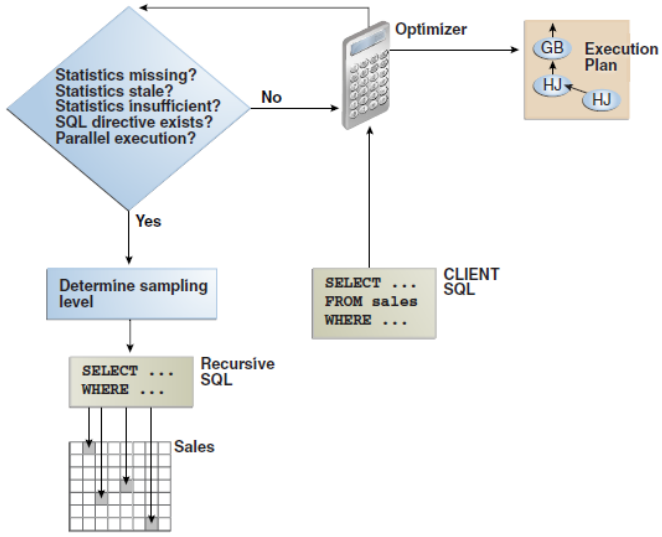
- Size of the table or index in database blocks
- Number of rows
- Average row size and chain count (tables only)
- Height and number of deleted leaf rows (indexes only)
- Number of distinct values for each column
- Number of distinct indexed values (indexes only)

As data is inserted, deleted, and modified, these statistics change. Because the performance impact of maintaining real-time data distribution statistics is prohibitive, these statistics are updated by periodically gathering statistics on tables and indexes.

Optimizer statistics are collected automatically by an automatic maintenance job that runs during predefined maintenance windows once daily by default. System statistics are operating system characteristics that are used by the optimizer. These statistics are not collected automatically. For details about collecting system statistics, see the *Oracle Database Performance Tuning Guide*.

Optimizer statistics are not the same as the database performance statistics that are gathered in the Automatic Workload Repository (AWR) snapshot.

Optimizer Statistics Collection



Optimizer statistics are collections of data that are specific details about database objects. These statistics are essential for the query optimizer to choose the best execution plan for each SQL statement. These statistics are gathered periodically and do not change between gatherings.

Statistics can be collected in the following ways:

- Automatically: Automatic Maintenance Tasks
- Manually: `DBMS_STATS` package
- By setting database initialization parameters
- By importing statistics from another database
- By configuring high-frequency automatic optimizer statistics collection to complement the standard statistics collection job

The recommended approach to gathering optimizer statistics is to allow the Oracle Database server to automatically gather the statistics. Automatic Maintenance Tasks can be created automatically at database creation time and are managed by the Scheduler. They gather statistics on all objects in the database that have either missing or stale optimizer statistics by default. You can change the default configuration through the Automatic Maintenance Tasks page.

System statistics describe the system's hardware characteristics, such as I/O and CPU performance and utilization, to the query optimizer. When choosing an execution plan, the optimizer estimates the I/O and CPU resources required for each query. System statistics enable the query optimizer to more accurately estimate I/O and CPU costs and, thereby, choose a better execution plan. System statistics are collected by using the `DBMS_STATS.GATHER_SYSTEM_STATS` procedure. When the Oracle Database server gathers system statistics, it analyzes system activity in a specified period of time. System statistics are not automatically gathered. Oracle Corporation recommends that you use the `DBMS_STATS` package to gather system statistics.

- If you choose not to use automatic statistics gathering, you must manually collect statistics in all schemas, including system schemas. If the data in your database changes regularly, you also need to gather statistics regularly to ensure that the statistics accurately represent characteristics of your database objects. To manually collect statistics, use the `DBMS_STATS` package. This PL/SQL package is also used to modify, view, export, import, and delete statistics.
- You can also manage optimizer and system statistics collection through database initialization parameters. For example:
  - The `OPTIMIZER_DYNAMIC_SAMPLING` parameter controls the level of dynamic sampling performed by the optimizer. You can use dynamic sampling to estimate statistics for tables and relevant indexes when they are not available or are too out of date to trust. Dynamic sampling also estimates single-table predicate selectivity when collected statistics cannot be used or are likely to lead to significant errors in estimation.
  - The `STATISTICS_LEVEL` parameter controls all major statistics collections or advisories in the database and sets the statistics collection level for the database. The values for this parameter are `BASIC`, `TYPICAL`, and `ALL`. You can query the `V$STATISTICS_LEVEL` view to determine which parameters are affected by the `STATISTICS_LEVEL` parameter.

  Note: Setting `STATISTICS_LEVEL` to `BASIC` disables many automatic features and is not recommended.

Setting Optimizer Statistics Preferences

DBMS_STATS.GATHER_*_STATS procedures: Gather statistics for an entire database or for individual objects using default values.

Use the SET_*_PREFS procedures to create preference values for any object that is not owned by SYS or SYSTEM.

Query DBA_TAB_STAT_PREFS to view object-level preferences.

Execute the DBMS_STATS.GET_PREFS procedure for each preference to see the global preferences.

The DBMS_STATS.GATHER_*_STATS procedures can be called at various levels to gather statistics for an entire database or for individual objects, such as tables. When the GATHER_*_STATS procedures are called, several of the parameters are often allowed to default. The supplied defaults work well for most of the objects in the database, but for some objects or schemas, the defaults need to be changed. Instead of running manual jobs for each of these objects, Oracle Database enables you to set values (called preferences) for individual objects, schemas, or databases or change the default values with a global-level command.

The preferences specify the parameters that are given to the gather procedures. The SET_*_PREFS procedures create preference values for any object that is not owned by SYS or SYSTEM. The expected use is that the DBA will set the global preferences for any parameters that should be databasewide. These will be applied to any parameter that is allowed to default.

The SET_DATATBASE_PREFS procedure iterates over all the tables and schemas in the database, setting the specified preference. SET_SCHEMA_PREFS iterates over the tables in the specified schema. SET_TABLE_PREFS sets the preference value for a single table.

All object preferences—whether set at the database, schema, or table level—are held in a single table. Changing preferences at the schema level overwrites the preferences that were previously set at the table level.

When the various gather procedures execute, they retrieve the object-level preferences that were set for each object. You can view the object-level preferences in the DBA_TAB_STAT_PREFS view. Any preferences that are not set at the object level will be set to the global level. You can see the global preferences by calling the DBMS_STATS.GET_PREFS procedure for each preference.

For details about these preferences, see the DBMS_STATS documentation in the *Oracle Database PL/SQL Packages and Types Reference*.

Preferences may be deleted with the DBMS_STATS.DELETE_*_PREFS procedures at the table, schema, and database levels. You can reset the global preferences to the recommended values with the DBMS_STATS.RESET_PARAM_DEFAULTS procedure.

Optimizer Statistics Advisor

- If best practices change in a new release, Optimizer Statistics Advisor encodes these practices in its rules.
- The advisor always provides the most up-to-date recommendations.
- Track and analyze how statistics are collected.
  - Class of findings: System, Operations, Objects
- Scope of findings
  - Problems with gathering of statistics
  - Status of automatic statistic gathering jobs
  - Quality of current statistics
- Suggestion for changes to the statistics collection

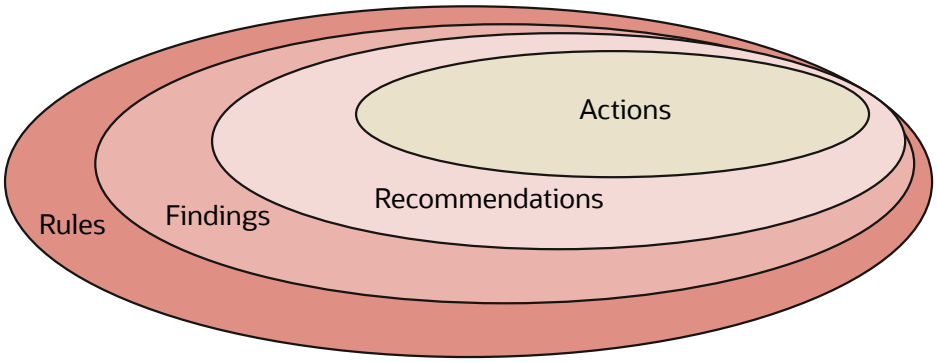
Since the introduction of the cost-based optimizer, optimizer statistics play a significant part in determining the execution plan for queries. Therefore, it is critical for the optimizer to have accurate and up-to-date statistics. The `DBMS_STATS` package serves this purpose and is improved in every release by adding new features. However, under many circumstances, these new features have not been fully utilized by customers or are being used in incorrect ways. Customers often use scripts and settings from one release to the next, based on earlier experience. These settings and methods may have been superseded or produce statistics that no longer give the most effective optimizer results.

Optimizer Statistics Advisor uses rules consistent with the current release to recommend changes to the way statistics are being gathered.

The advisor has a set of rules or recommended practices that are compared against the current statistics to generate findings. The rules are applied at the system, operation, or object level, such as whether the Automatic Gather Statistics jobs are scheduled, the statistics gathering procedures are using default parameters, and statistics are consistent across related objects. These rules check on issues related to the gathering of statistics—the schedules, parameters, and errors related to the automatic statistics gathering jobs. The rules include a variety of object-related issues, including whether the incremental mode setting is efficient.

Optimizer Statistics Advisor Report

- Report sections:
  - Header
  - Summary
  - Errors
  - Findings

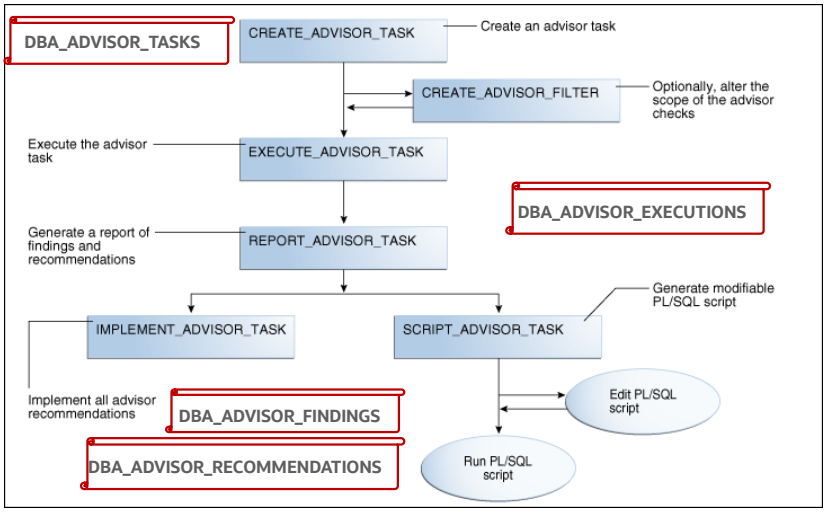


The Optimizer Statistics Advisor report has four basic concepts:

- **Rules:** Check the current configuration, history, and current statistics. Rules are added and changed by release to reflect best practices.
- **Findings:** They are generated when rules are not followed. An individual rule may generate many findings, but each finding is generated by only one rule. Some findings may be informational only, such as object staleness.
- **Recommendations:** They are responses the customer could make to resolve the finding. It is possible that several recommendations could be generated, and further investigation would be required by the customer. One or more rationales are given for each recommendation. Not all findings generate recommendations.
- **Actions:** They are PL/SQL statements or commands that the user can simply run in the command line to solve problems. They are provided in the form of scripts. Not all recommendations generate actions. For some recommendations, it is not possible to generate an action.

The report has sections for header, summary, errors, and findings. The header includes the advisor task parameters. The summary lists findings, and the errors section lists any errors the task encountered. The findings section includes the rule, findings, recommendations, and actions for each rule that produces a finding.

Executing Optimizer Statistics Advisor Tasks



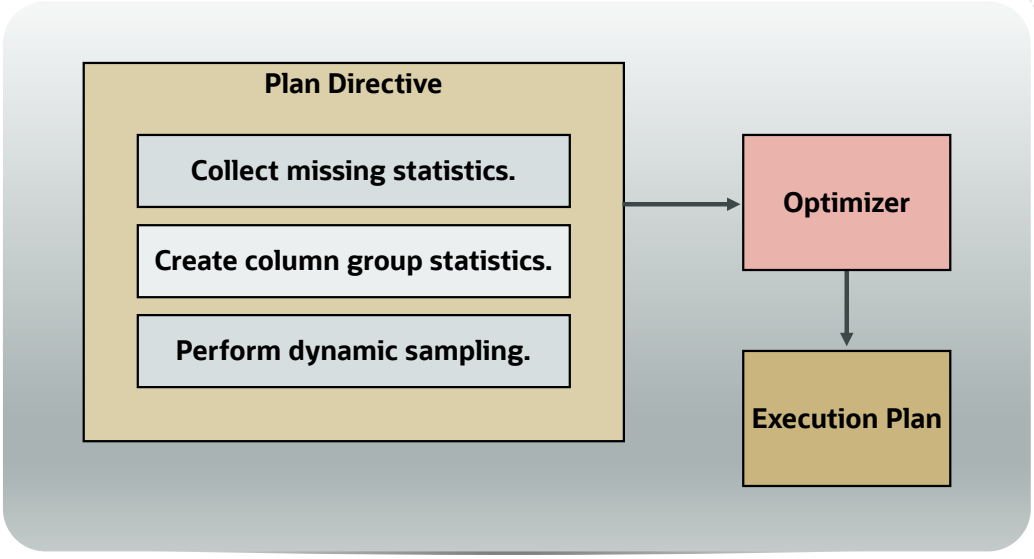
An Optimizer Statistics Advisor task can be executed with PL/SQL calls. Each task must be provided a unique task name. The definitions of the `CREATE_ADVISOR_TASK()` function parameters are:

- `TASK_NAME`: Name of the Statistics Advisor task
- `TIME_LIMIT`: The maximum duration the task can run

A filter list can be applied to the task to limit the scope of an advisor task using inclusion or exclusion lists for a user-specified set of rules, schemas, or operations. System rules are always checked. For example, you can configure an advisor task to include only recommendations for the SH schema. Also, you could exclude all violations of the rule for stale statistics.

You can create filters with the following `DBMS_STATS` procedures either individually or in combination: `CONFIGURE_ADVISOR_OBJ_FILTER`, `CONFIGURE_ADVISOR_RULE_FILTER`, and `CONFIGURE_ADVISOR_OPR_FILTER`. An Optimizer Statistics Advisor report can be generated with PL/SQL calls. The `REPORT_ADVISOR_TASK` function produces a report in text, HTML, or XML format. `IMPLEMENT_ADVISOR_TASK` implements the recommendations of the task based on the filters in place. An additional parameter, `LEVEL`, can be set to either `TYPICAL` or `ALL`. `TYPICAL` is the default. `ALL` ignores the filters.

SQL Plan Directives



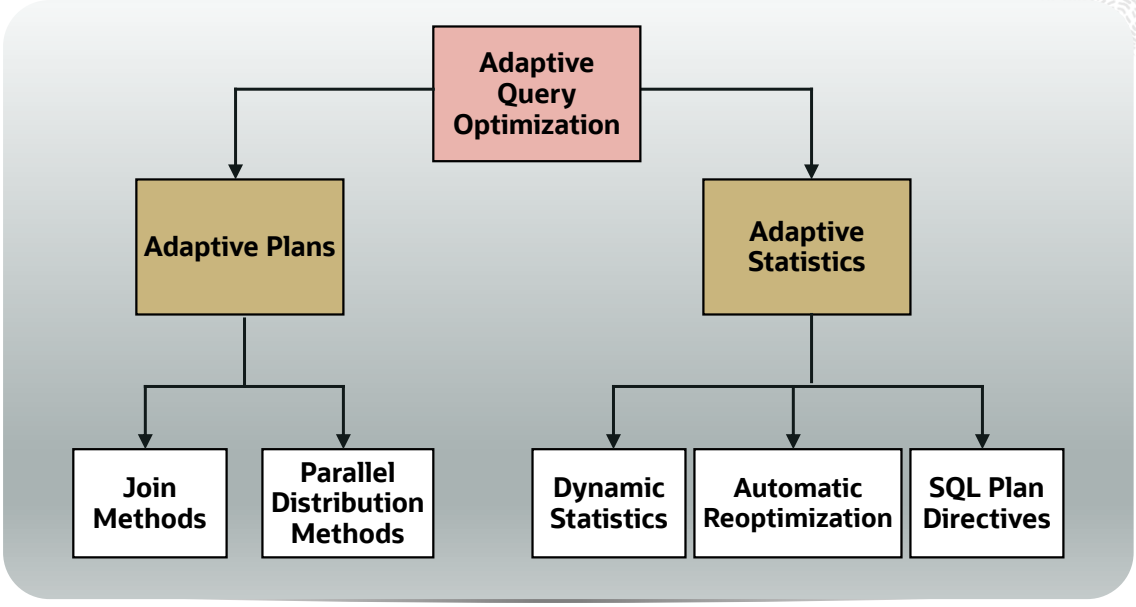
The Oracle Database server can use a SQL plan directive, which is additional information and instructions that the optimizer can use to generate a more optimal plan. For example, a SQL plan directive might instruct the optimizer to collect missing statistics, create column group statistics, or perform dynamic sampling. During SQL compilation or execution, the database analyzes the query expressions that are missing statistics or that misestimate optimizer cardinality to create SQL plan directives. When the optimizer generates an execution plan, the directives give the optimizer additional information about objects that are referenced in the plan.

SQL plan directives are not tied to a specific SQL statement or SQL ID. The optimizer can use SQL plan directives for SQL statements that are nearly identical because SQL plan directives are defined on a query expression. For example, directives can help the optimizer with queries that use similar patterns, such as web-based queries that are the same except for a select list item. The database stores SQL plan directives persistently in the `SYSAUX` tablespace. When generating an execution plan, the optimizer can use SQL plan directives to obtain more information about the objects that are accessed in the plan.

Directives are automatically maintained, created as needed, and purged if not used after a year.

Directives can be monitored in `DBA_SQL_PLAN_DIR_OBJECTS`. SQL plan directives improve plan accuracy by persisting both compilation and execution statistics in the `SYSAUX` tablespace, allowing them to be used by multiple SQL statements.

Adaptive Execution Plans



The Adaptive Execution Plans feature enables the optimizer to automatically adapt a poorly performing execution plan at run time and prevent a poor plan from being chosen on subsequent executions. The optimizer instruments its chosen plan so that at run time, it can be detected if the optimizer's estimates are not optimal. Then the plan can be automatically adapted to the actual conditions. An adaptive plan is a plan that changes after optimization when optimizer estimates prove inaccurate.

The optimizer can adapt plans based on statistics that are collected during statement execution. All adaptive mechanisms can execute a plan that differs from the plan that was originally determined during hard parse. This improves the ability of the query-processing engine (compilation and execution) to generate better execution plans.

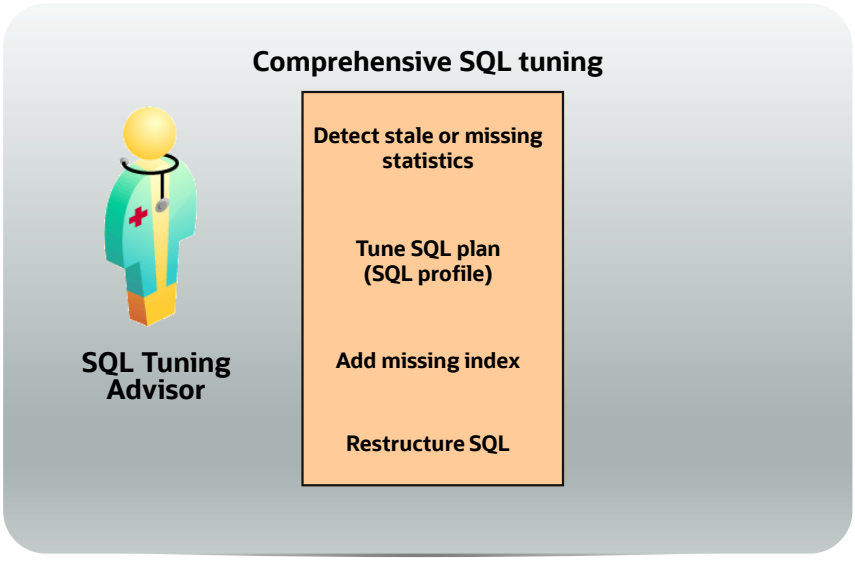
The two Adaptive Execution Plan techniques are:

- **Dynamic plans:** A dynamic plan chooses among subplans during statement execution. For dynamic plans, the optimizer must decide which subplans to include in a dynamic plan, which statistics to collect to choose a subplan, and thresholds for this choice.
- **Re-optimization:** In contrast, re-optimization changes a plan for executions after the current execution. For re-optimization, the optimizer must decide which statistics to collect at which points in a plan and when re-optimization is feasible.

Note: `OPTIMIZER_ADAPTIVE_REPORTING_ONLY` controls reporting-only mode for adaptive optimizations. When set to `TRUE`, adaptive optimizations run in reporting-only mode where the information required for an adaptive optimization is gathered, but no action is taken to change the plan.

The optimizer can pick the best-performing plan during any execution of the statement, not just the first execution. If the underlying data changes, or if queries re-execute with different input data, then the optimizer can adapt its plan to match the statistics gathered in the current execution. The continuous adaptive query plan adapts for every execution of the same cursor instead of only once.

SQL Tuning Advisor: Overview



SQL Tuning Advisor is the primary driver of the tuning process. It performs several types of analyses:

- **Statistics Analysis:** Checks each query object for missing or stale statistics and makes recommendations to gather relevant statistics
- **SQL Profiling:** The optimizer verifies its own estimates and collects auxiliary information to remove estimation errors. It builds a SQL profile by using the auxiliary information and makes a recommendation to create it. When a SQL profile is created, it enables the query optimizer to generate a well-tuned plan.
- **Access Path Analysis:** New indexes are considered if they significantly improve access to each table in the query. When appropriate, recommendations to create such objects are made.
- **SQL Structure Analysis:** SQL statements that use bad plans are identified and relevant suggestions are made to restructure them. The suggested changes can be syntactic as well as semantic.

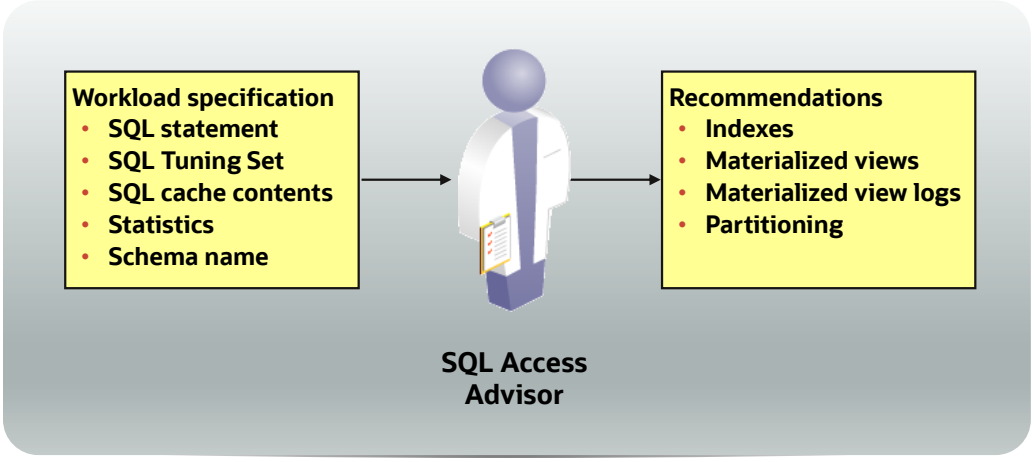
SQL Tuning Advisor considers each SQL statement included in the advisor task independently. Creating a new index may help a query, but may hurt the response time of DML. So, a recommended index or other object should be checked with SQL Access Advisor over a workload (a set of SQL statements) to determine whether there is a net gain in performance.

SQL Tuning Advisor runs automatically every night as the Automatic SQL Tuning Task. There may be times when a SQL statement needs immediate tuning action. You can use SQL Tuning Advisor to analyze SQL statements and obtain performance recommendations at any time. Typically, you run this advisor as an ADDM performance-finding action.

Additionally, you can run SQL Tuning Advisor when you want to analyze the top SQL statements consuming the most CPU time, I/O, and memory.

Even though you can submit multiple statements to be analyzed in a single task, each statement is analyzed independently. To obtain tuning recommendations that consider the overall performance of a set of SQL, use SQL Access Advisor.

SQL Access Advisor: Overview



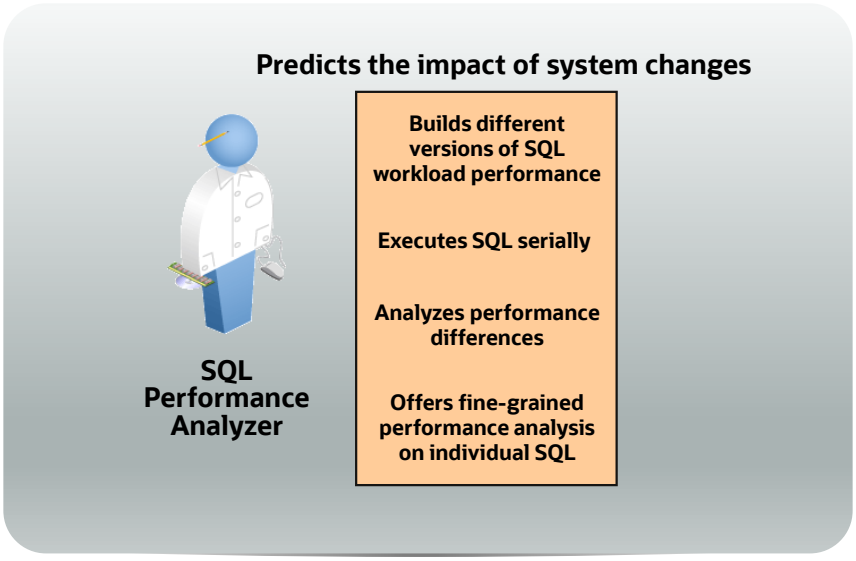
SQL Access Advisor can recommend the proper set of materialized views, materialized view logs, partitioning, and indexes for a given workload. Understanding and using these structures is essential when optimizing SQL because they can result in significant performance improvements in data retrieval.

SQL Access Advisor recommends bitmap, function-based, and B-tree indexes. A bitmap index offers a reduced response time for many types of ad hoc queries and reduced storage requirements compared to other indexing techniques. B-tree indexes are most commonly used in a data warehouse to index unique or near-unique keys.

Another component of SQL Access Advisor also recommends how to optimize materialized views so that they can be fast refreshable and take advantage of general query rewrite.

Note: For more information about materialized views and query rewrite, see *Oracle Database Performance Tuning Guide*.

SQL Performance Analyzer: Overview



Oracle Database includes SQL Performance Analyzer, which gives you an exact and accurate assessment of the impact of change on the SQL statements that make up the workload. SQL Performance Analyzer helps you forecast the impact of a potential change on the performance of a SQL query workload. This capability provides you with detailed information about the performance of SQL statements, such as before-and-after execution statistics, and statements with performance improvement or degradation. This enables you (for example) to make changes in a test environment to determine whether the workload performance will be improved through a database upgrade.

SQL Performance Analyzer includes the following capabilities:

- Helps predict the impact of system changes on SQL workload response time
- Builds different versions of SQL workload performance (that is, SQL execution plans and execution statistics)
- Executes SQL serially (concurrency not honored)
- Analyzes performance differences
- Offers fine-grained performance analysis on individual SQL
- Is integrated with SQL Tuning Advisor to tune regressions

Use Cases

SQL Performance Analyzer can be used to predict and prevent potential performance problems for any database environment change that affects the structure of SQL execution plans. The changes can include (but are not limited to) any of the following:

- Database upgrades
- Implementation of tuning recommendations
- Schema changes
- Statistics gathering
- Database parameter changes
- OS and hardware changes

You can use SQL Performance Analyzer to predict SQL performance changes that result from changes for even the most complex environments. As applications evolve through the development life cycle, database application developers can test changes to schemas, database objects, and rewritten applications to mitigate any potential performance impact.

SQL Performance Analyzer also enables the comparison of SQL performance statistics.

You can access SQL Performance Analyzer through Enterprise Manager or by using the `DBMS_SQLPA` package.

For details about the `DBMS_SQLPA` package, see the *Oracle Database PL/SQL Packages and Types Reference Guide*.

Managing Automated Tuning Tasks

- Use the DBMS_AUTO_TASK_ADMIN.ENABLE procedure to manage automatic space and performance tuning tasks.
- Set the CLIENT_NAME parameter to the following values based on the task:
  - Automatic statistics collection: auto optimizer stats collection
  - Automatic SQL Tuning task: sql tuning advisor
  - Segment shrink: auto space advisor

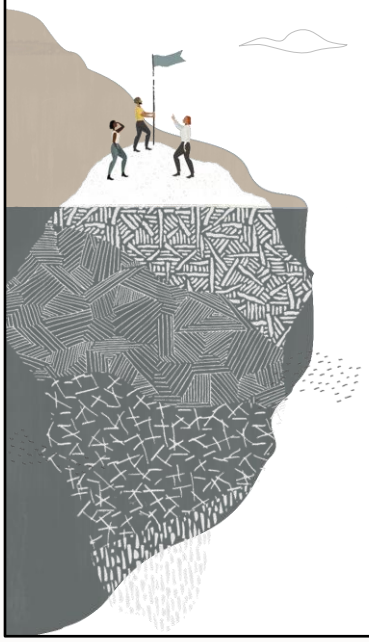
```
BEGIN
  DBMS_AUTO_TASK_ADMIN.ENABLE (
     client_name => 'auto optimizer stats collection',
     operation   => NULL, window_name => NULL);
END;
```

To control the automatic segment shrink task, the automatic statistics collection, or the SQL Tuning Advisor automated task, use the DBMS_AUTO_TASK_ADMIN.ENABLE procedure as shown in this example:

```
BEGIN
  DBMS_AUTO_TASK_ADMIN.ENABLE (
     client_name => 'auto optimizer stats collection',
     operation   => NULL, window_name => NULL);
END;
```

See the *Oracle Database SQL Tuning Guide* for additional examples.

Summary



- Describe the SQL tuning methodology
- Manage optimizer statistics
- Use SQL Tuning Advisor to identify and tune SQL statements that are using the most resources
- Use SQL Access Advisor to tune a workload