7

Managing the Database Instance

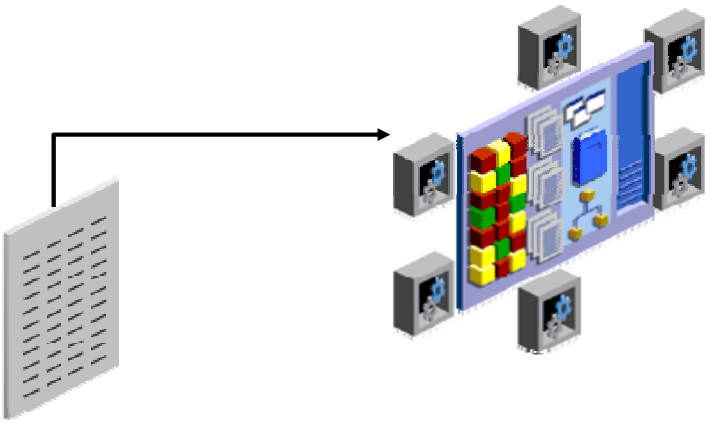
ORACLE

Objectives

After completing this lesson, you should be able to:

- Start and stop the Oracle database instance and components
- Modify database initialization parameters
- Describe the stages of database startup
- Describe database shutdown options
- View the alert log
- Access dynamic performance views

Initialization Parameter Files



spfileorcl.ora

or

initorcl.ora

When you start the instance, an initialization parameter file is read. There are two types of parameter files:

- **Server parameter file (SPFILE):** This is the preferred type of initialization parameter file. It is a binary file that can be written to and read by the database server and *must not be edited manually*. It resides on the server on which the Oracle instance is executing; it is persistent across shutdown and startup. The default name of this file, which is automatically sought at startup, is `spfile<SID>.ora`.
- **Text initialization parameter file:** This type of initialization parameter file can be read by the database server, but it is not written to by the server. The initialization parameter settings must be set and changed manually by using a text editor so that they are persistent across shutdown and startup. The default name of this file (which is automatically sought at startup if an SPFILE is not found) is `init<SID>.ora`.

It is recommended that you create an SPFILE as a dynamic way to maintain initialization parameters.

Note: The Oracle Database server searches the `$ORACLE_HOME/dbs` directory on Linux for the initialization files.

Types of Values for Initialization Parameters

The Oracle Database server has the following types of values for initialization parameters:

- Boolean
- String
- Integer
- Parameter File
- Reserved
- Big Integer

Derived Parameter Values

Some initialization parameters are derived, meaning that their values are calculated from the values of other parameters. Normally, you should not 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.

Operating System–Dependent Parameter Values

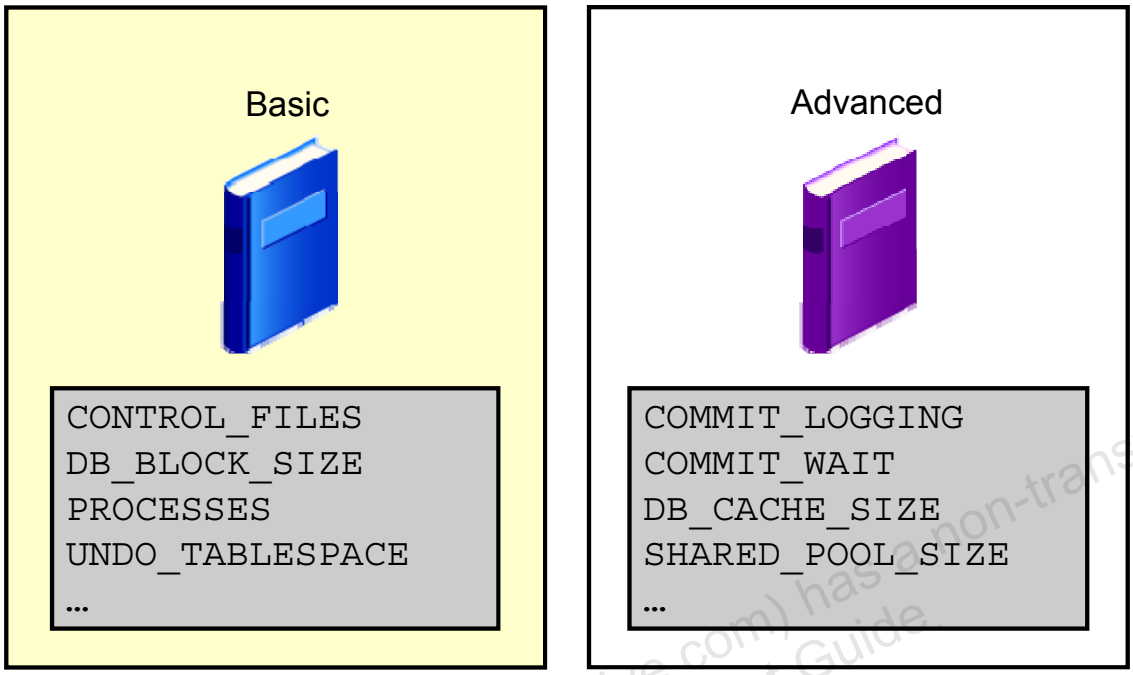
The valid values or value ranges of some initialization parameters depend on the host operating system. For example, the `DB_FILE_MULTIBLOCK_READ_COUNT` parameter specifies the maximum number of blocks that are read in one I/O operation during a sequential scan; this parameter is platform dependent.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

Setting Parameter Values

Initialization parameters offer the most potential for improving system performance. Some parameters set capacity limits but do not affect performance. For example, when the value of `OPEN_CURSORS` is `10`, a user process attempting to open its eleventh cursor receives an error. Other parameters affect performance but do not impose absolute limits. For example, reducing the value of `OPEN_CURSORS` does not prevent work even though it may slow performance.

Increasing the values of parameters may improve your system's performance, but increasing most parameters also increases the System Global Area (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. Operating system parameters that control virtual memory working areas should be set with the SGA size in mind. The operating system configuration can also limit the maximum size of the SGA.

Types of Initialization Parameters



Initialization parameters are of two types: basic and advanced.

In the majority of cases, it is necessary to set and tune only the 30 or so basic parameters to get reasonable performance from the database. In rare situations, modification of the advanced parameters may be needed to achieve optimal performance. There are more than 300 advanced parameters.

A basic parameter is defined as one that you are likely to set to keep your database running with good performance. All other parameters are considered to be advanced.

Examples of basic parameters:

- Determining the global database name: `DB_NAME` and `DB_DOMAIN`
- Specifying a fast recovery area and size: `DB_RECOVERY_FILE_DEST` and `DB_RECOVERY_FILE_DEST_SIZE`
- Specifying the total size of all SGA components: `SGA_TARGET`
- Specifying the method of undo space management tablespace: `UNDO_TABLESPACE`
- `COMPATIBLE` initialization parameter and irreversible compatibility

Note: Some of the initialization parameters are listed on the following pages. For a complete list, see the *Oracle Database Reference*.

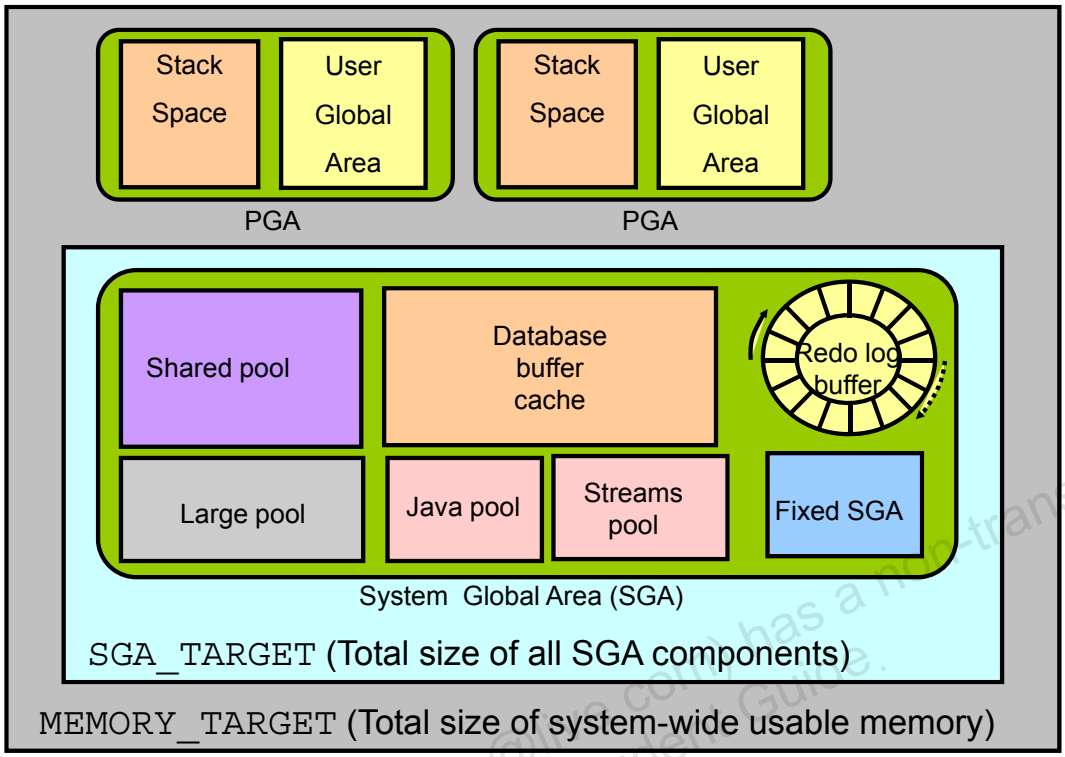
Initialization Parameters: Examples

Parameter	Specifies
CONTROL_FILES	One or more control file names
DB_FILES	Maximum number of database files
PROCESSES	Maximum number of OS user processes that can simultaneously connect
DB_BLOCK_SIZE	Standard database block size used by all tablespaces
DB_CACHE_SIZE	Size of the standard block buffer cache

ORACLE

- **CONTROL_FILES parameter:** Specifies one or more control file names. Oracle strongly recommends that you multiplex and mirror control files. Range of values: from one to eight file names (with path names). Default value: OS dependent.
- **DB_FILES parameter:** Specifies the maximum number of database files that can be opened for this database. Range of values: OS dependent. Default value: 200.
- **PROCESSES parameter:** Specifies the maximum number of OS user processes that can simultaneously connect to an Oracle server. This value should allow for all background processes and user processes. Range of values: from 6 to an OS-dependent value. Default value: Dynamic and dependent on the number of CPUs.
- **DB_BLOCK_SIZE parameter:** Specifies the size (in bytes) of an Oracle database block. This value is set at database creation and cannot be subsequently changed. This specifies the standard block size for the database. All tablespaces will use this size by default. Range of values: 2048 to 32768 (OS-dependent). Default value: 8192.
- **DB_CACHE_SIZE parameter:** Specifies the size of the default buffer pool. Range of values: At least 4 MB times the number of CPUs (smaller values are automatically rounded up to this value). Default value: 0 if SGA_TARGET is set, otherwise, the larger of 48 MB or (4 MB*CPU_COUNT).

Initialization Parameters: Examples



`SGA_TARGET` specifies the total size of all SGA components. If `SGA_TARGET` is specified, the following memory pools are automatically sized:

- Buffer cache (`DB_CACHE_SIZE`)
- Shared pool (`SHARED_POOL_SIZE`)
- Large pool (`LARGE_POOL_SIZE`)
- Java pool (`JAVA_POOL_SIZE`)
- Streams pool (`STREAMS_POOL_SIZE`)

If these automatically tuned memory pools are set to nonzero values, the values are used as minimum levels by Automatic Shared Memory Management (ASMM). You set minimum values if an application component needs a minimum amount of memory to function properly.

The following pools are manually sized components and are not affected by ASMM:

- Log buffer
- Other buffer caches (such as `KEEP` and `RECYCLE`) and other block sizes
- Fixed SGA and other internal allocations

The memory allocated to these pools is deducted from the total available memory for `SGA_TARGET` when ASMM is enabled.

Note: The MMON process computes the values of the automatically tuned memory pools to support ASMM.

`MEMORY_TARGET` specifies the Oracle systemwide usable memory. The database tunes memory to the `MEMORY_TARGET` value, reducing or enlarging the SGA and PGA as needed.

In a text-based initialization parameter file, if you omit `MEMORY_MAX_TARGET` and include a value for `MEMORY_TARGET`, the database automatically sets `MEMORY_MAX_TARGET` to the value of `MEMORY_TARGET`. If you omit the line for `MEMORY_TARGET` and include a value for `MEMORY_MAX_TARGET`, the `MEMORY_TARGET` parameter defaults to zero. After startup, you can then dynamically change `MEMORY_TARGET` to a nonzero value if it does not exceed the value of `MEMORY_MAX_TARGET`. The `MEMORY_TARGET` parameter is modifiable with the `ALTER SYSTEM` command. Values range from 152 MB to `MEMORY_MAX_TARGET`.

Initialization Parameters: Examples

Parameter	Specifies
PGA_AGGREGATE_TARGET	Amount of PGA memory available to all server processes
SHARED_POOL_SIZE	Size of shared pool (in bytes)
UNDO_MANAGEMENT	Undo space management mode to be used

ORACLE

- **PGA_AGGREGATE_TARGET parameter:** Specifies the amount of Program Global Area (PGA) memory available to all server processes attached to the instance. This memory does not reside in the System Global Area (SGA). The database uses this parameter as a target amount of PGA memory to use. When setting this parameter, subtract the SGA from the total memory on the system that is available to the Oracle instance. The minimum value is 10 MB and the maximum value is (4096 GB – 1). The default is 10 MB or 20% of the size of the SGA, whichever is greater.
- **SHARED_POOL_SIZE parameter:** Specifies the size of the shared pool in bytes. The shared pool contains objects such as shared cursors, stored procedures, control structures, and parallel execution message buffers. Range of values: OS-dependent. Default value: 0 if SGA_TARGET is set, otherwise 128 MB if 64-bit; 48 MB if 32-bit.
- **UNDO_MANAGEMENT parameter:** Specifies the undo space management mode that the system should use. When set to AUTO, the instance is started in automatic undo management mode. Otherwise, it is started in rollback undo mode. In rollback undo mode, undo space is allocated as rollback segments. In automatic undo mode, undo space is allocated as undo tablespaces. Range of values: AUTO or MANUAL. If the UNDO_MANAGEMENT parameter is omitted when the instance is started, the default value AUTO is used.

Using SQL*Plus to View Parameters

```
SQL> SELECT name, value FROM V$PARAMETER;
NAME                 VALUE
------------         ----------
lock_name_space
processes             300
sessions              472
timed_statistics      TRUE
timed_os_statistics   0
…

SQL> SHOW PARAMETER SHARED_POOL_SIZE
NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------------
securefile_log_shared_pool_size      big integer 0
shared_pool_size                     big integer 0

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
…
```


The slide shows examples of using SQL*Plus to view parameters. You can query the V$PARAMETER view to find the values of the various parameters. V$PARAMETER displays the current parameter values in the current session. You can also use the SHOW PARAMETER command with any string to view parameters that contain that string.

The query in the following example is requesting the name and values of the parameters. Use a WHERE clause to specify specific parameter names:

```
SQL> SELECT name, value FROM V$PARAMETER
  2  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
…
10 rows selected.
```

Description of the view:

```
SQL> desc V$parameter
 Name                                      Null?  Type
 ----------------------------------------- ------- -------------
 NUM                                               NUMBER
 NAME                                              VARCHAR2(80)
 TYPE                                              NUMBER
 VALUE                                             VARCHAR2(4000)
 DISPLAY_VALUE                                     VARCHAR2(4000)
 ISDEFAULT                                         VARCHAR2(9)
 ISSES_MODIFIABLE                                  VARCHAR2(5)
 ISSYS_MODIFIABLE                                  VARCHAR2(9)
 ISPDB_MODIFIABLE                                  VARCHAR2(5)
 ISINSTANCE_MODIFIABLE                             VARCHAR2(5)
 ISMODIFIED                                        VARCHAR2(10)
 ISADJUSTED                                        VARCHAR2(5)
 ISDEPRECATED                                      VARCHAR2(5)
 ISBASIC                                           VARCHAR2(5)
 DESCRIPTION                                       VARCHAR2(255)
 UPDATE_COMMENT                                    VARCHAR2(255)
 HASH                                              NUMBER
 CON_ID                                            NUMBER
```

The second example shows the use of the SQL*Plus `SHOW PARAMETER` command to view parameter settings. You can also use this command to find all parameters that contain a text string. For example, you can find all parameter names that include the string `db` by using the following command:

```
SQL> show parameter db
NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------
…
db_8k_cache_size                     big integer 0
db_big_table_cache_percent_target    string      0
db_block_buffers                     integer     0
db_block_checking                    string      FALSE
db_block_checksum                    string      TYPICAL
…
```

Other Views Containing Information About Parameters

- `V$SPPARAMETER`: Displays information about the contents of the server parameter file. If a server parameter file was not used to start the instance, each row of the view will contain `FALSE` in the `ISSPECIFIED` column.
- `V$PARAMETER2`: Displays information about the initialization 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 instance-wide values displayed in the `V$SYSTEM_PARAMETER2` view.
- `V$SYSTEM_PARAMETER`: Displays information about the initialization parameters that are currently in effect for the instance.

Changing Initialization Parameter Values

- Static parameters:
  - Can be changed only in the parameter file
  - Require restarting the instance before taking effect
- Dynamic parameters:
  - Can be changed while database is online
  - Can be altered at:
    - Session level
    - System level
  - Are valid for duration of session or based on `SCOPE` setting
  - Are changed by using `ALTER SESSION` and `ALTER SYSTEM` commands

There are two types of initialization parameters.

Static parameters: Affect the instance or entire database and can be modified only by changing the contents of the text initialization parameter file or the server parameter file. Static parameters require the database to be shut down and restarted to take effect. They cannot be changed for the current instance.

Dynamic parameters: Can be changed while the database is online. There are two types:

- *Session-level parameters* affect only a user session. Examples include national language support (NLS) parameters that can be used to specify national language settings for sorts, date parameters, and so on. You can use these in a given session; they expire when the session ends.
- *System-level parameters* affect the entire database and all sessions. Examples include modifying the `SGA_TARGET` value and setting archive log destinations. These parameters stay in effect based on the `SCOPE` specification. To make them permanent, you have to add these parameter settings to the server parameter file by specifying the `SCOPE=both` option or manually editing the text initialization parameter file.

Dynamic parameters can be changed by using the `ALTER SESSION` and `ALTER SYSTEM` commands.

Use the SET clause of the ALTER SYSTEM statement to set or change initialization parameter values. The optional SCOPE clause specifies the scope of a change as follows:

- **SCOPE=SPFILE:** The change is applied in the server parameter file only. No change is made to the current instance. For both dynamic and static parameters, the change is effective at the next startup and is persistent. This is the only SCOPE specification allowed for static parameters.
- **SCOPE=MEMORY:** The change is applied in memory only. The change is made to the current instance and is effective immediately. For dynamic parameters, the effect is immediate but not persistent because the server parameter file is not updated. For static parameters, this specification is not allowed.
- **SCOPE=BOTH:** The change is applied in both the server parameter file and memory. The change is made to the current instance and is effective immediately. For dynamic parameters, the effect is persistent because the server parameter file is updated. For static parameters, this specification is not allowed.

You must not specify SCOPE=SPFILE or SCOPE=BOTH if the instance did not start up with a server parameter file. The default is SCOPE=BOTH if a server parameter file was used to start up the instance, and the default is MEMORY if a text initialization parameter file was used to start up the instance.

For some dynamic parameters, you can also specify the DEFERRED keyword. When it is specified, the change is effective only for future sessions. This is valid for only the following parameters:

- AUDIT_FILE_DEST
- BACKUP_TAPE_IO_SLAVES
- OBJECT_CACHE_MAX_SIZE_PERCENT
- OBJECT_CACHE_OPTIMAL_SIZE
- OLAP_PAGE_POOL_SIZE
- RECYCLEBIN
- SORT_AREA_RETAINED_SIZE
- SORT_AREA_SIZE

When you specify SCOPE as SPFILE or as BOTH, an optional COMMENT clause lets you associate a text string with the parameter update. The comment is written to the server parameter file.

Changing Parameter Values: Examples

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

Session altered.

SQL> SELECT SYSDATE FROM dual;

SYSDATE
-----------
oct 17 2012
```

```
SQL> ALTER SYSTEM  SET
  2  SEC_MAX_FAILED_LOGIN_ATTEMPTS=2
  3  COMMENT='Reduce for tighter security.'
  4  SCOPE=SPFILE;

System altered.
```


The first statement in the slide is an example of changing a session-level parameter. The user is setting the session date format to be `mon dd yyyy`. As a result, any queries on the date will display dates in that format. Session-level parameters can also be set in applications by using PL/SQL.

The second statement changes the maximum number of failed login attempts before the connection is dropped. It includes a comment and explicitly states that the change is to be made only in the server parameter file. After the specified number of failure attempts, the connection is automatically dropped by the server process. This is not a dynamic parameter and the Oracle database instance will need to be restarted before the change can take effect.

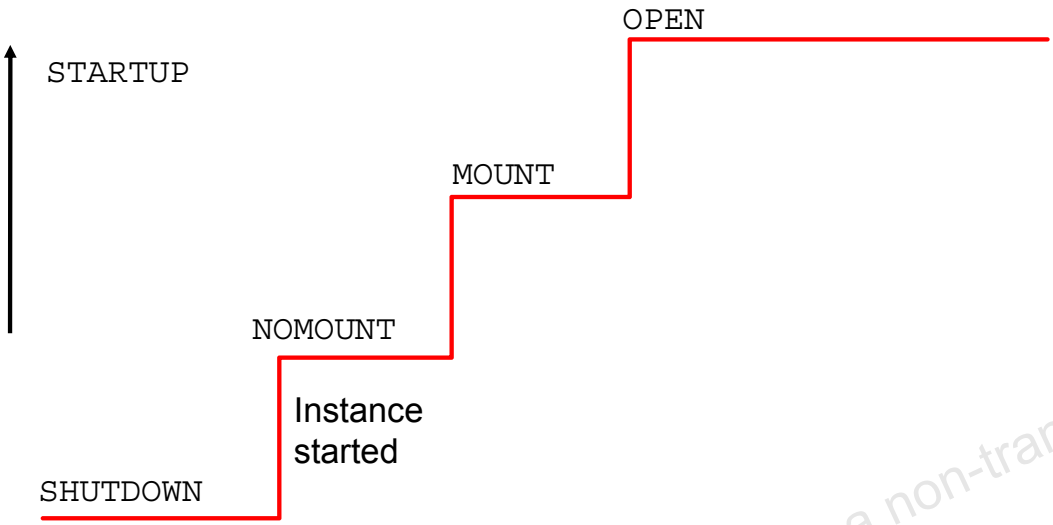
Quiz

The majority of database parameters are dynamic and can be changed without having to shut down the database instance.

a. True
b. False

Answer: a

Starting Up an Oracle Database Instance: NOMOUNT



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.

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.

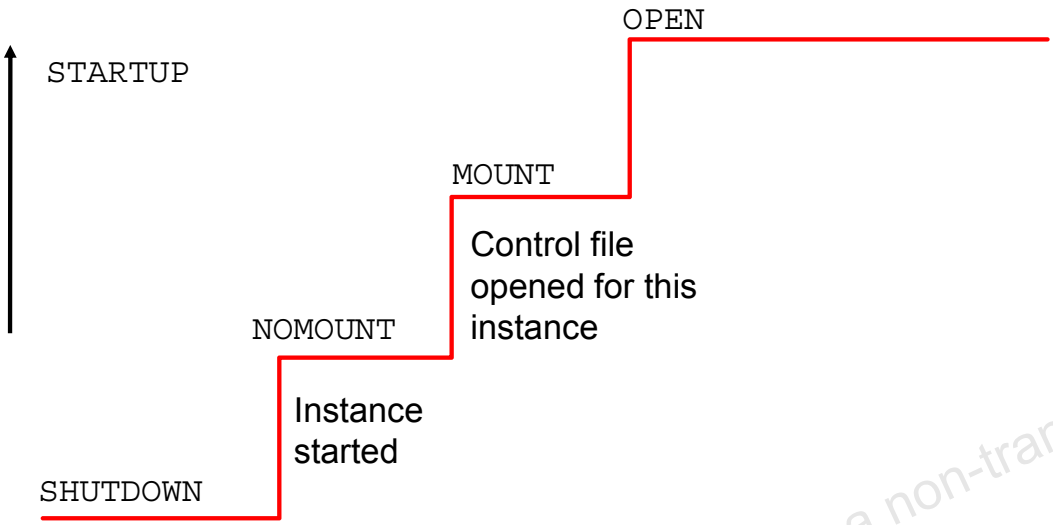
When an instance is started, the following takes place:

- Searching `$ORACLE_HOME/dbs` for a file of a particular name in this sequence:
  1. Search for `spfile<SID>.ora`.
  2. If `spfile<SID>.ora` is not found, search for `spfile.ora`.
  3. If `spfile.ora` is not found, search for `init<SID>.ora`.

     This is the file that contains initialization parameters for the instance. Specifying the `PFILE` parameter with `STARTUP` overrides the default behavior.
- Allocating the SGA
- Starting the background processes
- Opening the `alert_<SID>.log` file and the trace files

Note: `SID` is the system ID, which identifies the instance name (for example, `ORCL`).

Starting Up an Oracle Database Instance: MOUNT



Mounting a database includes the following:

- Associating a database with a previously started instance
- Locating and opening all the control files specified in the parameter file
- Reading the control files to obtain the names and statuses of the data files and online redo log files. (However, no checks are performed to verify the existence of the data files and online redo log files at this time.)

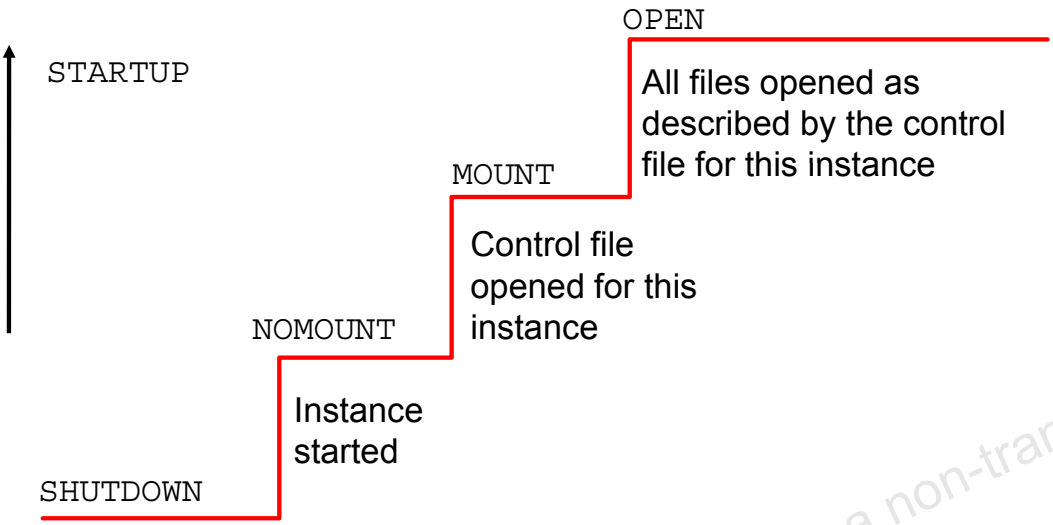
To perform specific maintenance operations, start an instance and mount a database, but do not open the database.

For example, the database must be mounted but must not be opened during the following tasks:

- Renaming data files. (Data files for an offline tablespace can be renamed when the database is open.)
- Enabling and disabling online redo log file archiving options
- Performing full database recovery

Note: A database may be left in `MOUNT` mode even though an `OPEN` request has been made. This may be because the database needs to be recovered in some way. If recovery is performed while in the `MOUNT` state, the redo logs are open for reads and the data files are open as well to read the blocks needing recovery and to write blocks if required during recovery.

Starting Up an Oracle Database Instance: OPEN



A normal database operation means that an instance is started and the database is mounted and opened. With a normal database operation, any valid user can connect to the database and perform typical data access operations.

Opening the database includes the following:

- Opening the data files
- Opening the online redo log files

If any of the data files or online redo log files are not present when you attempt to open the database, the Oracle server returns an error.

During this final stage, the Oracle server verifies that all data files and online redo log files can be opened, and checks the consistency of the database. If necessary, the System Monitor (SMON) background process initiates instance recovery.

You can start up a database instance in restricted mode so that only Oracle Database users with the `RESTRICTED SESSION` system privilege can connect to the database.

Startup Options: Examples

- Using the SQL*Plus utility:

```
SQL> startup                          1
```

```
SQL> startup nomount                  2
```

```
SQL> alter database mount;            3
```

```
SQL> alter database open;             4
```

- Using the Server Control utility with Oracle Restart

```
$ srvctl start database -d orcl -o mount
```


The slide shows the SQL*Plus syntax to start up the database.

1. This command starts the instance, associates the database files to it, and mounts and opens the database.
2. This command starts the instance and the database is not mounted.
3. This command mounts a database from the `NOMOUNT` state.
4. This command opens the database from the `MOUNT` state.

When the database is enabled with Oracle Restart, the Server Control (SRVCTL) utility can be used to start the database instance. The SRVCTL utility has the advantage that it can also start all required dependent resources such as the ASM instance, ASM disk groups, and listener.

Shutdown Modes

Shutdown Modes	A	I	T	N
Allows new connections	No	No	No	No
Waits until current sessions end	No	No	No	Yes
Waits until current transactions end	No	No	Yes	Yes
Forces a checkpoint and closes files	No	Yes	Yes	Yes

Shutdown modes:

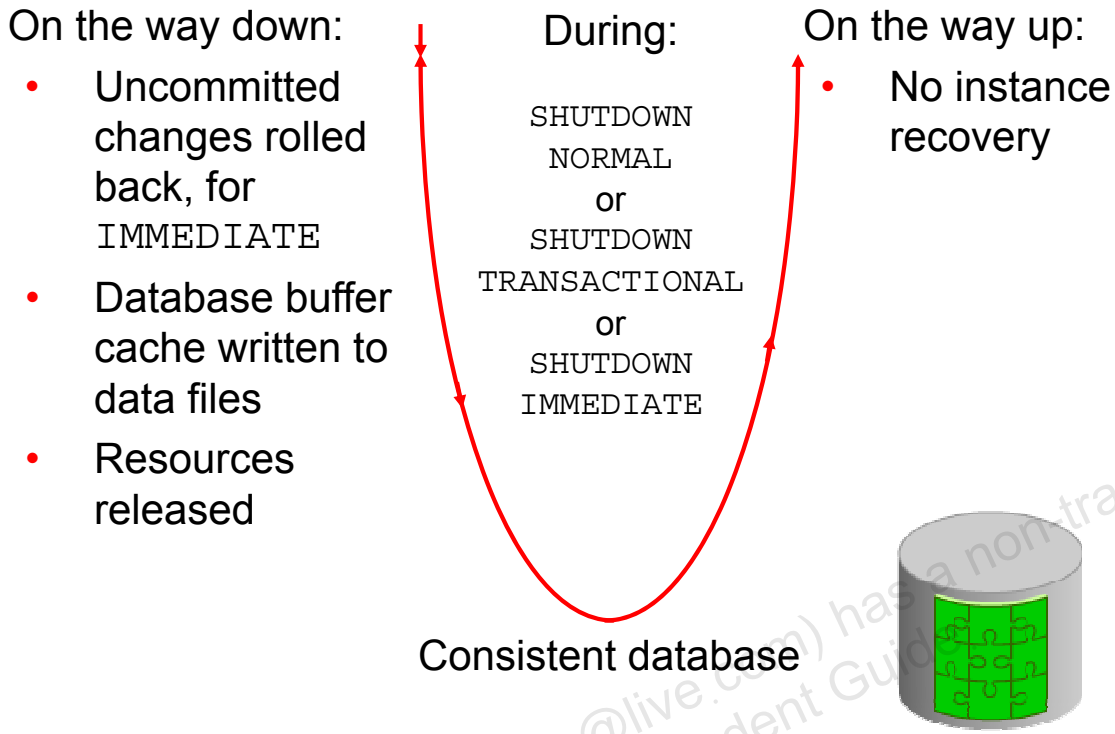
- A = ABORT
- I = IMMEDIATE
- T = TRANSACTIONAL
- N = NORMAL

Shutdown modes are progressively more accommodating of current activity in this order:

- **ABORT:** Performs the least amount of work before shutting down. Because this mode requires recovery before startup, use it only when necessary. It is typically used when no other form of shutdown works, when there are problems with starting the instance, or when you need to shut down immediately because of an impending situation (such as notice of a power outage within seconds).
- **IMMEDIATE:** Is the most typically used option. Uncommitted transactions are rolled back.
- **TRANSACTIONAL:** Allows existing transactions to finish, but does not allow new transactions to start
- **NORMAL:** Waits for sessions to disconnect

If you consider the amount of time that it takes to perform the shutdown, you find that ABORT is the fastest and NORMAL is the slowest. NORMAL and TRANSACTIONAL can take a long time depending on the number of sessions and transactions.

Shutdown Options



SHUTDOWN NORMAL

NORMAL is the default shutdown mode if no mode is specified. A normal database shutdown proceeds with the following conditions:

- 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.
- The next startup does not require an instance recovery.

SHUTDOWN TRANSACTIONAL

A shutdown in TRANSACTIONAL mode prevents clients from losing data, including results from their current activity. A transactional database shutdown proceeds with the following conditions:

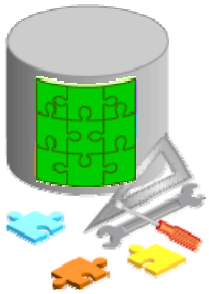
- No client can start a new transaction on this particular instance.
- A client is disconnected when the client ends the transaction that is in progress.
- When all transactions have been completed, a shutdown occurs immediately.
- The next startup does not require an instance recovery.

SHUTDOWN IMMEDIATE

A shutdown in `IMMEDIATE` mode proceeds with the following conditions:

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

Shutdown Options



On the way down:

- Modified buffers not written to data files
- Uncommitted changes not rolled back

During:

`SHUTDOWN ABORT`
or
Instance failure
or
`STARTUP FORCE`

Inconsistent database

On the way up:

- Online redo log files used to reapply changes
- Undo segments used to roll back uncommitted changes
- Resources released

ORACLE

`SHUTDOWN ABORT`

If shutdown in `NORMAL`, `TRANSACTIONAL`, and `IMMEDIATE` modes does not work, you can abort the current database instance. Aborting an instance proceeds with the following conditions:

- 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.

Note: It is not advisable to back up a database that is in an inconsistent state.

Shutdown Options: Examples

- Using SQL*Plus:

```
SQL> shutdown                          1
```

```
SQL> shutdown transactional            2
```

```
SQL> shutdown immediate                3
```

```
SQL> shutdown abort                    4
```

- Using the SRVCTL utility with Oracle Restart

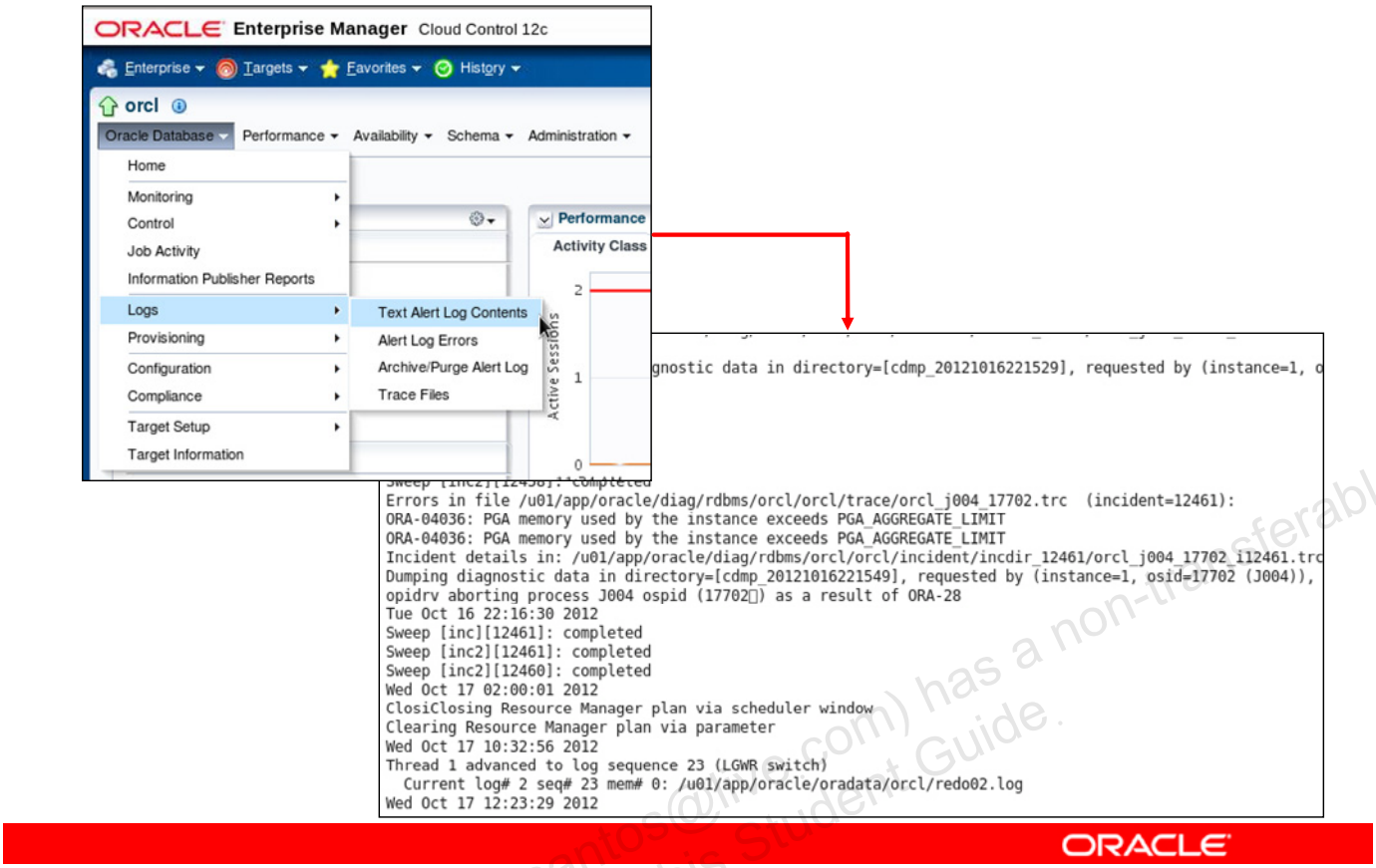
```
$ srvctl stop database -d orcl -o abort
```


The slide shows examples using both SQL*Plus and the SRVCTL utility to shut down the database.

1. This command initiates a normal shutdown. The database will not shut down until all users have logged out.
2. This command initiates a transactional shutdown. The database will not shut down until all existing transactions are completed.
3. This command initiates an immediate shutdown. Uncommitted transactions will be rolled back.
4. This command initiates a shutdown abort.

When the database is enabled with Oracle Restart, the SRVCTL utility can be used to shut down the database instance.

Viewing the Alert Log



Each database 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.

The alert file of a database is a chronological log of messages such as the following:

- 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

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.

To determine the location of the alert log with SQL*Plus:

- Connect to the database with SQL*Plus (or another query tool such as SQL Developer).
- Query the `V$DIAG_INFO` view.

To view the text-only alert log without the XML tags:

- In the `V$DIAG_INFO` query results, note the path that corresponds to the Diag Trace entry. Change to the location specified.
- Open the `alert_SID.log` file with a text editor.

To view the XML-formatted alert log:

- In the `V$DIAG_INFO` query results, note the path that corresponds to the Diag Alert entry. Change directory to that path.
- Open the `log.xml` file with a text editor.

Using Trace Files

- Each server and background process can write to an associated trace file.
- Error information is written to the corresponding trace file.
- Automatic diagnostic repository (ADR)
  - Is a systemwide central tracing and logging repository
  - Stores database diagnostic data such as:
    - Traces
    - Alert log
    - Health monitor reports

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.

The ADR is a systemwide 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 *ADR base*. Its location is set by the `DIAGNOSTIC_DEST` initialization parameter.

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

```
./diag/product_type/db_id/instance_id
```

Administering the DDL Log File

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

```
$ more ddl_orcl.log
Thu Nov 15 08:35:47 2012
diag_adl:drop user app_user
```


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. Refer to the *Oracle Database Reference* for a complete list of DDL commands that are captured in the DDL log.

Locate the log files as follows:

```
$ pwd
/u01/app/oracle/diag/rdbms/orcl/orcl/log
$ ls
ddl  ddl_orcl.log  debug  test
$ cd ddl
$ ls
log.xml
```

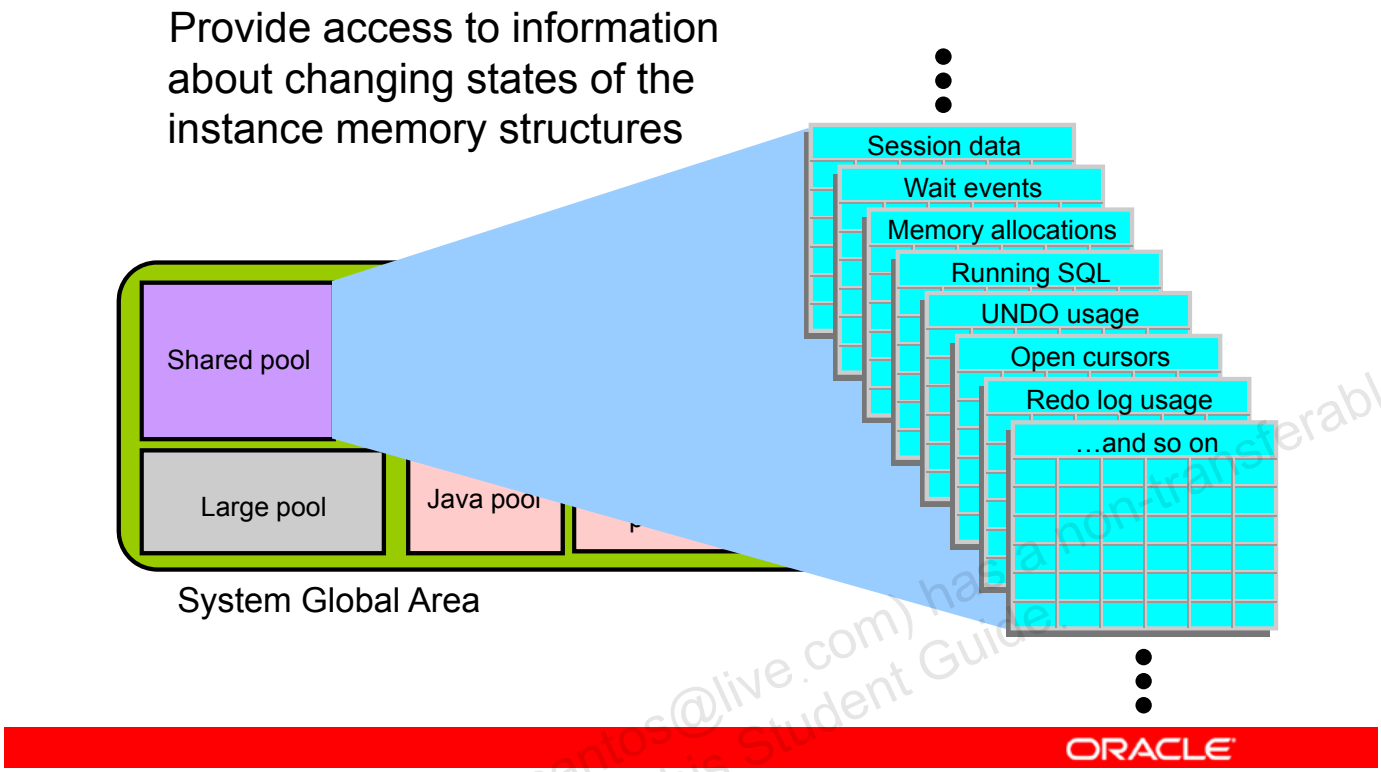
Understanding the Debug Log File

- Debug log contains warnings about conditions, states, or events that do not inhibit correct operation of an Oracle Database component
- The log is intended for use by Oracle Support when diagnosing a problem.
- It is included in incident packaging service (IPS) incident packages.
- It is written to `$ORACLE_BASE/diag/rdbms/<db_name>/<SID>/debug`.

The debug log contains warnings generated by Oracle Database components. The warnings are about conditions, states, or events that may be out of the ordinary but do prevent the component from operating correctly. For this reason, these warnings are not written to the alert log. If they are needed in the future to diagnose a problem, they are recorded in the debug log.

Typically, the debug log would be needed only by Oracle Support to diagnose a problem, so you do not need to view the contents of the debug log on a regular basis. The debug log is also included in incident packaging service (IPS) incident packages. IPS is discussed in detail in a later lesson.

Using Dynamic Performance Views



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

Dynamic performance views include information about:

- Sessions
- File states
- Progress of jobs and tasks
- Locks
- Backup status
- Memory usage and allocation
- System and session parameters
- SQL execution
- Statistics and metrics

Note: The `DICT` and `DICT_COLUMNS` views also contain the names of these dynamic performance views. Dynamic performance views start with the prefix `v$`.

Dynamic Performance Views: Usage Examples

1
```
SELECT sql_text, executions FROM v$sql
WHERE cpu_time > 200000;
```

2
```
SELECT * FROM v$session
WHERE machine = 'EDXX9P1'
AND logon_time > SYSDATE - 1;
```

3
```
SELECT sid, ctime FROM v$lock
WHERE block > 0;
```


The three examples shown in the slide answer the following questions:

1. For which SQL statements (and their associated numbers of executions) is the CPU time consumed greater than 200,000 microseconds?
2. Which current sessions have logged in from the `EDXX9P1` computer in the last day?
3. 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?

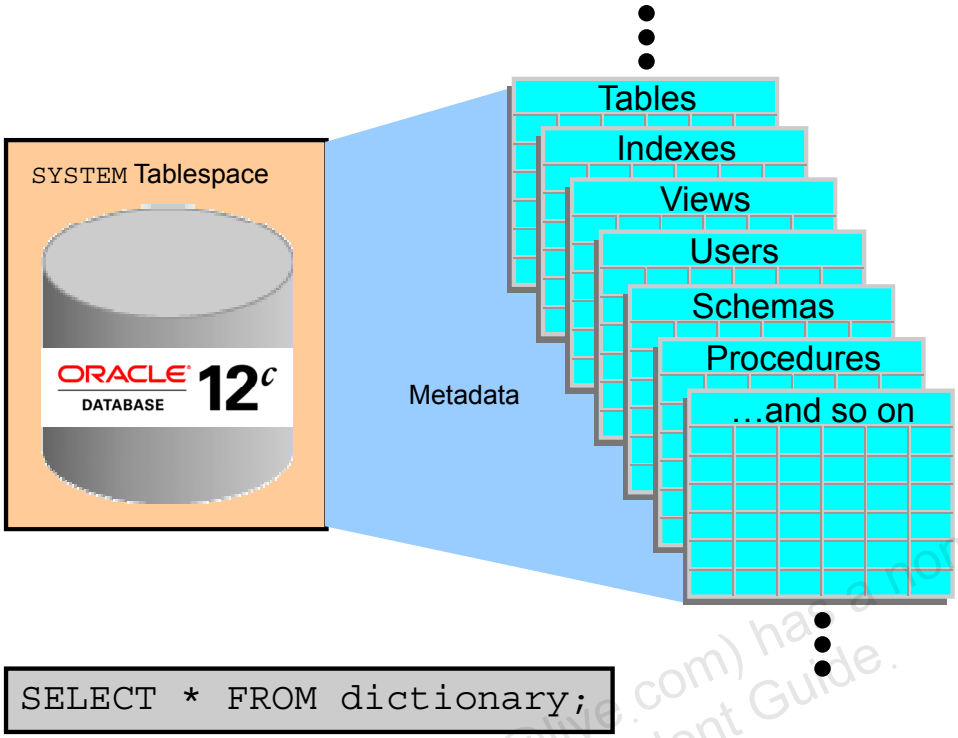
Dynamic Performance Views: Considerations

- These views are owned by the SYS user.
- Different views are available at different times:
  - The instance has been started.
  - The database is mounted.
  - The database is 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 contain data 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$BGPROCESS to see the list of background processes that are running. But you cannot query V$DATAFILE to see the status of database data files because it is the mounting of a database that reads the control file to find out 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 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 as 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 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.

Data Dictionary Views

	Who Can Query	Contents	Subset of	Notes
DBA_	DBA	Everything	N/A	May have additional columns meant for DBA use only
ALL_	Everyone	Everything that the user has privileges to see	DBA_ views	Includes user's own objects and other objects that the user has been granted privileges to see
USER_	Everyone	Everything that the user owns	ALL_ views	Is usually the same as ALL_ except for the missing OWNER column. (Some views have abbreviated names as PUBLIC synonyms.)

The view prefixes indicate the data (and how much of that data) a given user can see.

A global view of everything is accessed only by users with DBA privileges, using the DBA_ prefix.

The next level of privilege is at the ALL_ prefix level, which represents all objects that the querying 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.

The USER_ prefix represents the smallest scope of visibility. This type of view shows only those objects that the querying user owns (that is, those that are present in the user's own schema).

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. This is dependent on the nature of the information in the view. For example, there is a DBA_LOCK view, but there is no ALL_LOCK view. This is 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 DBA_ views can be queried only by users with the SYSDBA, SELECT ANY DICTIONARY, or SELECT_CATALOG_ROLE privilege.

Not all dictionary views start with the prefix DBA_, ALL_, and USER_. The following views or synonyms to views are exceptions to this:

- AUDIT_ACTIONS
- CAT
- CHANGE_PROPAGATIONS
- CHANGE_PROPAGATION_SETS
- CHANGE_SETS
- CHANGE_SOURCES
- CHANGE_TABLES
- CLIENT_RESULT_CACHE_STATS$
- CLU
- COLS
- COLUMN_PRIVILEGES
- DATABASE_COMPATIBLE_LEVEL
- DBMS_ALERT_INFO
- DBMS_LOCK_ALLOCATED
- DICT
- DICTIONARY
- DICT_COLUMNS
- DUAL
- GLOBAL_NAME
- IND
- INDEX_HISTOGRAM
- INDEX_STATS
- LOGSTDBY_UNSUPPORTED_TABLES
- NLS_DATABASE_PARAMETERS
- NLS_INSTANCE_PARAMETERS
- NLS_SESSION_PARAMETERS
- OBJ
- RECYCLEBIN
- RESOURCE_COST
- ROLE_ROLE_PRIVS
- ROLE_SYS_PRIVS
- ROLE_TAB_PRIVS
- SEQ
- SESSION_PRIVS
- SESSION_ROLES

- SM$VERSION
- SYN
- TABLE_PRIVILEGES
- TABS

Data Dictionary: Usage Examples

1
```
SELECT table_name, tablespace_name
FROM user_tables;
```

2
```
SELECT sequence_name, min_value, max_value,
increment_by
FROM all_sequences
WHERE sequence_owner IN ('MDSYS','XDB');
```

3
```
SELECT USERNAME, ACCOUNT_STATUS
FROM dba_users
WHERE ACCOUNT_STATUS = 'OPEN';
```

4
```
DESCRIBE dba_indexes
```


The example queries in the slide answer the following questions:

1. What are the names of the tables (along with the name of the tablespace where they reside) that have been created in your schema?
2. What is the significant information about the sequences in the database that you have access to?
3. What users in this database are currently able to log in?
4. What are the columns of the `DBA_INDEXES` view? This shows you what information you can view about all the indexes in the database. The following is a partial output of this command:

```
SQL> DESCRIBE dba_indexes
 Name              Null?    Type
 ----------------- -------- -------------
 OWNER             NOT NULL VARCHAR2(30)
 INDEX_NAME        NOT NULL VARCHAR2(30)
 INDEX_TYPE                 VARCHAR2(27)
 TABLE_OWNER       NOT NULL VARCHAR2(30)
 TABLE_NAME        NOT NULL VARCHAR2(30)
```

Quiz

Which data dictionary view can be used to find the names of all tables in the database?

a. USER_TABLES
b. ALL_TABLES
c. DBA_TABLES
d. ANY_TABLES

Answer: c

Summary

In this lesson, you should have learned how to:

- Start and stop the Oracle database instance and components
- Modify database initialization parameters
- Describe the stages of database startup
- Describe database shutdown options
- View the alert log
- Access dynamic performance views

Practice 7

- 7-1: Managing the Oracle Instance by Using Oracle Enterprise Manager Cloud Control
- 7-2: Managing the Oracle Instance by Using Oracle Enterprise Manager Database Express
- 7-3: Managing the Oracle Instance by Using SQL*Plus
- 7-4: Viewing the Alert Log by Using the Automatic Diagnostic Repository Command Interface (ADRCI)