Architecture of Oracle Database Management System

Oracle is a Relational Database Management System (RDBMS), which uses Relational Data Model to store its database and SQL (commonly abbreviated as Structured Query Language) to process the stored data. The architecture of Oracle system can be best explained in terms of client/server paradigm. Thus, we will explain the architecture of Oracle server using the structure called ***instance.***

An oracle instance is a complex set of memory structures and operating system processes. It is the Oracle instance, which manages all database activities, such as transaction processing, database recovery, form generation, and so on. The instance structure is loosely styled after UNIX's implementation of multitasking operating system. Discrete processes perform specialized tasks within the RDBMS that work together to accomplish the goals of the ***instance.*** Each process has a separate memory block that it uses to store local and private variables, address stacks and other runtime information. The processes use a common shared memory area for processing data concurrently. This memory block is called the System Global Area (SGA). Figure 1 illustrates the architecture of an ***instance***. Each component of the instance is described below.

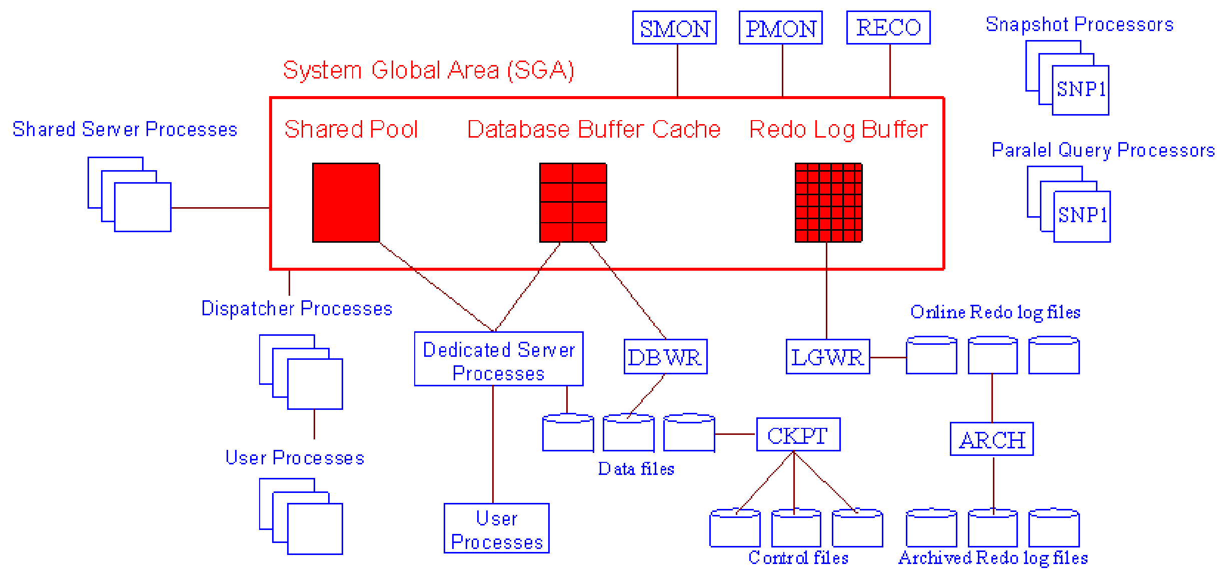


Figure 1: Oracle *instance* architecture

System Global Area (SGA)

The SGA is the primary memory component of the ***instance.*** It provides memory structure necessary for data manipulation, SQL statement parsing, and redo caching. The SGA is shared, which means that the multiple processes can access and modify the data contained in it in a synchronized manner. The SGA consists of the following components:

- Shared Pool
- Database buffer cache

- Redo Log Buffer
- Multithread server (MTS) structures.

The Shared Pool

Figure 2 illustrates the structure of a shared pool. It contains the library cache, the dictionary cache, and server control structures (such as the database character set).

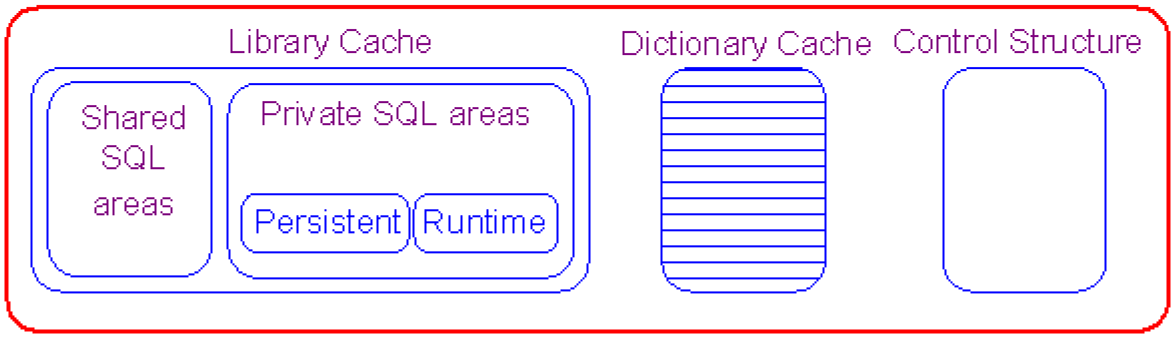


Figure 2: Oracle *Shared pool* architecture

Library cache: Stores the text, parsed format, and execution plan of SQL statements that have been submitted to the RDBMS, as well as the headers of PL/SQL packages and procedures that have been executed. For each SQL statement the server first checks the library cache to see if an identical statement has already been submitted and cached. If it has, then the server uses the stored parse tree and execution path for the statement, rather than building these structures from scratch.

The library cache has ***shared*** and ***private*** SQL areas. The ***shared*** area contains the parse tree and execution path for SQL statement. The ***private*** SQL area contains session specific information, such as bind variable, environment and session parameters, runtime stacks and buffers, etc. A ***private*** SQL area is created for each transaction initiated, and it is de-allocated after the cursor corresponding to that ***private*** area is closed. Using these two structures, the server can reuse the information common across all execution of an SQL statement, while session specific information to the execution can be retrieved from the ***private*** SQL area.

The ***private*** SQL area is further divided into ***persistent*** and ***runtime*** areas. The *persistent* area contains information that is valid and applicable through multiple executions of the SQL statement. The ***runtime*** area contains data that is used only while the SQL statement is being executed.

Dictionary cache: Stores data dictionary rows that have been used to parse SQL statements. Information such as segment information, security and access privileges, and available free storage space is held in this area.

The Database Buffer Cache

The buffer cache is composed of memory blocks. All data manipulated by Oracle server is first loaded into the buffer cache before being used. All data updates are performed in the buffer blocks

The data movement (swapping and loading) between buffer and disk or other parts of RAM is by least recently Used (LRU) algorithm. The LRU list keeps track of what data blocks are accessed and how often.

Buffer blocks that have been modified are called dirty and are placed on the dirty list. The dirty list keeps track of all data modifications made to the cache data that have not been flushed to disk. When Oracle receives a request to change data, the data change is made to the blocks in the buffer cache and written to the redo log, and then the block is put on the dirty list. Subsequent access to this data reads the new value from the changed data in the buffer cache. Dirty data from the dirty list are written to the disk database under deferred update policy.

The Redo Log Buffer

The redo log buffer is used to store redo information in memory before it is flushed to the redo log files on the disk. It is circular buffer.

The Oracle Background Process

The Oracle server process transactions concurrently. Thus, at any time there may be hundreds of simultaneous users performing a number of different operations. To accomplish these tasks, the server divides the entire workload between a number of programs, each of which operates largely independently of one another and has a specific role to play. These programs are referred to as the *Oracle background processes*. The Oracle background processes are:

- **SMON (System Monitor) and PMON (Process Monitor)**: SMON is the process that performs automatic instance recovery. If the last database shutdown was not clean, SMON automatically rolls forward the operations that were complete but could not be installed in the database, and rolls back unfinished transactions. SMON process also manages certain database segments, reclaiming temporary segment space no longer in use, and automatically combining contiguous blocks of free space in the data files.
- **PMON:** is responsible for cleaning up terminated or failed processes, rolling back uncommitted transactions, releasing the locks held by disconnected processes, and freeing SGA resources held by failed processes. It also monitors the server and dispatcher processes, automatically starting them if they fail.
- **DBWR**: Database Writer process is responsible for writing the dirty blocks from the database buffer cache to the data files on the disk. The process waits until certain criteria are met, then reads the dirty list and writes a set of modified blocks in batch. In most installations, there is one DBWR process to handle all the write activity of the database. However, more than one DBWR process can be started if one is incapable of keeping up with the demands of the database.

- **LGWR**: Log Writer is the process that writes redo log entries from the redo log buffer in the SGA to the online log files. LGWR performs this write when a commit occurs, the inactivity timeout for LGWR is reached, the redo log buffer becomes one-third full, or DBWR completes a flush of the data buffer blocks at a checkpoint. LGWR also handles multiple user commits simultaneously, if one or more users issue a commit before LGWR has completed flushing the buffer on behalf of another user's commit. It is important to note that Oracle does not regard a transaction as being complete until LGWR has flushed the redo information from the redo buffer to the online redo logs. It is LGWR's successful writing of the redo log entries into the online redo logs, and not the changing of data in the data files, which returns a success code to the server process.
- **DISPATCHER PROCESSES (Dnnn):** The dispatcher process passes user requests to the SGA request queue and returns the server responses back to the correct user process.
- **ARCH**: The archiver process is responsible for copying full online redo logs to the archived redo log files. While the archiver is copying the redo log, no other processes can write to the log. This is important to keep in mind, because of the circular nature of the redo logs. If the database needs to switch redo logs but the archiver is still copying the next log in the sequence, all database activity halts until archiver finishes.
- **CKPT**: The Checkpoint Process, is an optional background process that performs the checkpoint tasks that LGWR would normally perform-namely updating the data file and control file headers with the current version information. This process reduces the amount of work on LGWR when there are frequent checkpoints occurring, frequent log switches, or many data files in the database.
- **RECO**: Recovery Process is responsible for recovering failed transactions. In distributed database systems. It is automatically started when the database is configured for distributed transactions. The RECO process operates with little or no DBA intervention when an in-doubt transaction occurs in a distributed system. The RECO process attempts to connect to the remote database and resolves the in-doubt transaction when a database connection is successful.
- **SNPn**: The Snapshot Process, handles the automatic refreshing of database snapshots and runs the database procedures scheduled through the database system’s job package.
- **LCKn**: The lock process is responsible for managing and coordinating the locks held by the individual instances. Each instance in the parallel server installation has 1-10 lock processes assigned, and each instance must have the same number. This process has no purpose in a non-parallel server environment.
- **Pnnn**: Parallel query processes are named Pnnn. These processes are involved in parallel index creations, table creations, and queries.

USER AND SERVER PROCESSES (Snnn)

Applications and utilities access the RDBMS through a user process. The user process connects to a server process, which can be dedicated to one user process or shared among many. The server process parses and executes SQL statements that are submitted to it and returns the result sets back to the user process. It is also the process that reads data blocks from the data files into the database buffer cache.

Each process is allocated a section of memory referred to as the ***Process Global Area (PGA)***. The contents of the PGA differ depending on what type of connection is made to the database. When a user process connects to the database via a dedicated server process, user session data, stack space, and cursor state information is stored in the PGA. The user session data consists of security and resource usage information; the stack space contains local variables specific to the user session; and the cursor state area contains runtime information for the cursor, including rows returned and cursor return codes. If, however, the user process connects through a shared server process, the session and cursor state information is stored within the SGA.

Role and Responsibility of Database Administrator

Installation and configuration: The DBA must install and customize the Oracle Database 11*g* software and any assorted programs that will run alongside and access the database.

Create datafiles and tablespaces: The DBA decides which application the data will reside in.

Create and manage accounts: The DBA sets up the accounts with the usernames and passwords that interact with the database.

Tuning: The DBA tweaks the environment that Oracle Database 11*g* operates in by adjusting initialization parameters using the system parameter file.

Configure backups: Alongside recovery testing, the DBA performs this activity to ensure the usability and integrity of system backups.

Work with developers: This is an ongoing process for DBAs, to ensure that the code they write is optimal and that they use the server’s resources as efficiently as possible.

Stay current: DBAs keep abreast of the emerging technology and are involved in scoping out future directions based on the enhancements delivered with new software releases.

Work with Oracle Support Services DBAs initiate service requests (SRs) to engage support engineers in problem-solving endeavors.

Maximize resource efficiency: The DBA must tune Oracle Database 11*g* so that applications can coexist with one another on the same server and share that machine’s resources efficiently.

Recovery: The DBA is responsible in keeping database in the consistent state using the recovery mechanism.

Tablespaces and Datafiles

Oracle stores data logically in *tablespaces* and physically in *datafiles* associated with the corresponding tablespaces.

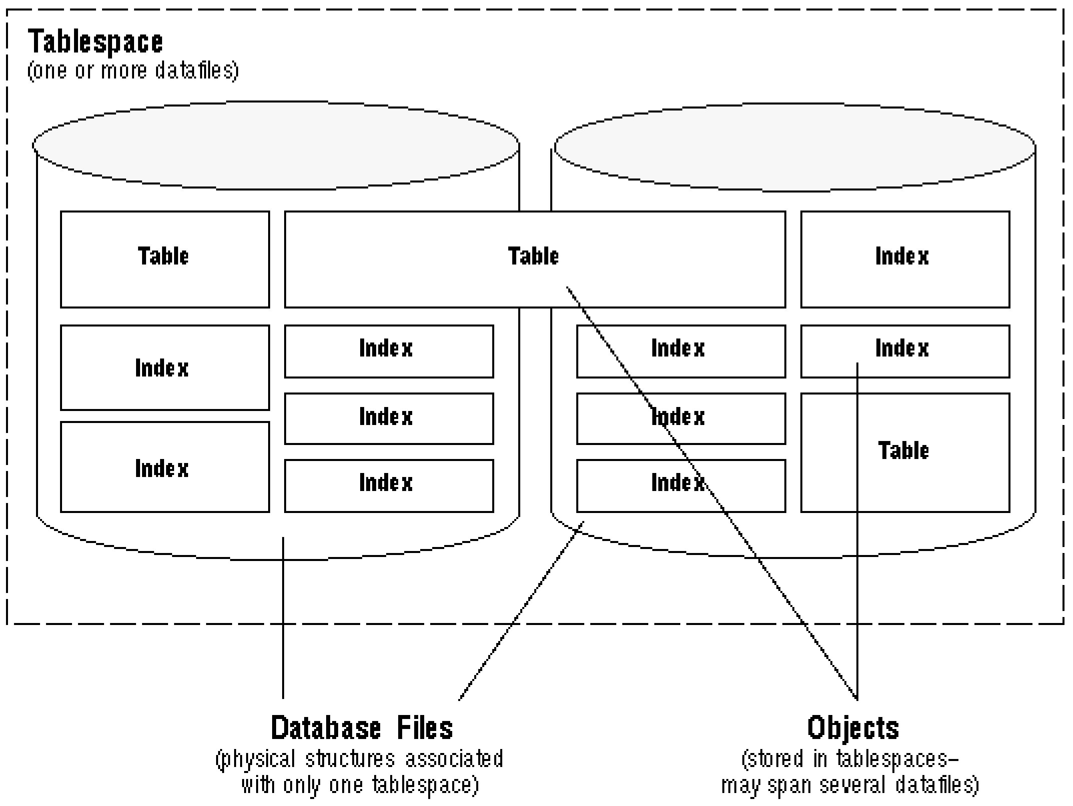


Figure: Datafiles and Tablespaces

Although databases, tablespaces, datafiles, and segments are closely related, they have important differences:

An Oracle database is comprised of one or more logical storage units called *tablespaces*. The database's data is collectively stored in the database's tablespaces.

Each tablespace in an Oracle database is comprised of one or more operating system files called *datafiles*. A tablespace's datafiles physically store the associated database data on disk.

A database's data is collectively stored in the datafiles that constitute each tablespace of the database. For example, the simplest Oracle database would have one tablespace and one datafile. A more complicated database might have three tablespaces, each comprised of two datafiles (for a total of six datafiles).

When a schema object such as a table or index is created, its segment is created within a designated tablespace in the database. For example, suppose you create a table in a specific tablespace using the CREATE TABLE command with the TABLESPACE option. Oracle allocates the space for this table's data segment in one or more of the datafiles that constitute the specified tablespace. An object's segment allocates space in only one tablespace of a database.

Tablespaces

A database is divided into one or more logical storage units called *tablespaces*. A database administrator can use tablespaces to do the following:

- control disk space allocation for database data
- assign specific space quotas for database users
- control availability of data by taking individual tablespaces online or offline
- perform partial database backup or recovery operations
- allocate data storage across devices to improve performance

A database administrator can create new tablespaces, add and remove datafiles from tablespaces, set and alter default segment storage settings for segments created in a tablespace, make a tablespace read-only or writeable, make a tablespace temporary or permanent, and drop tablespaces.

The SYSTEM Tablespace

Every Oracle database contains a tablespace named SYSTEM that Oracle creates automatically when the database is created. The SYSTEM tablespace always contains the data dictionary tables for the entire database.

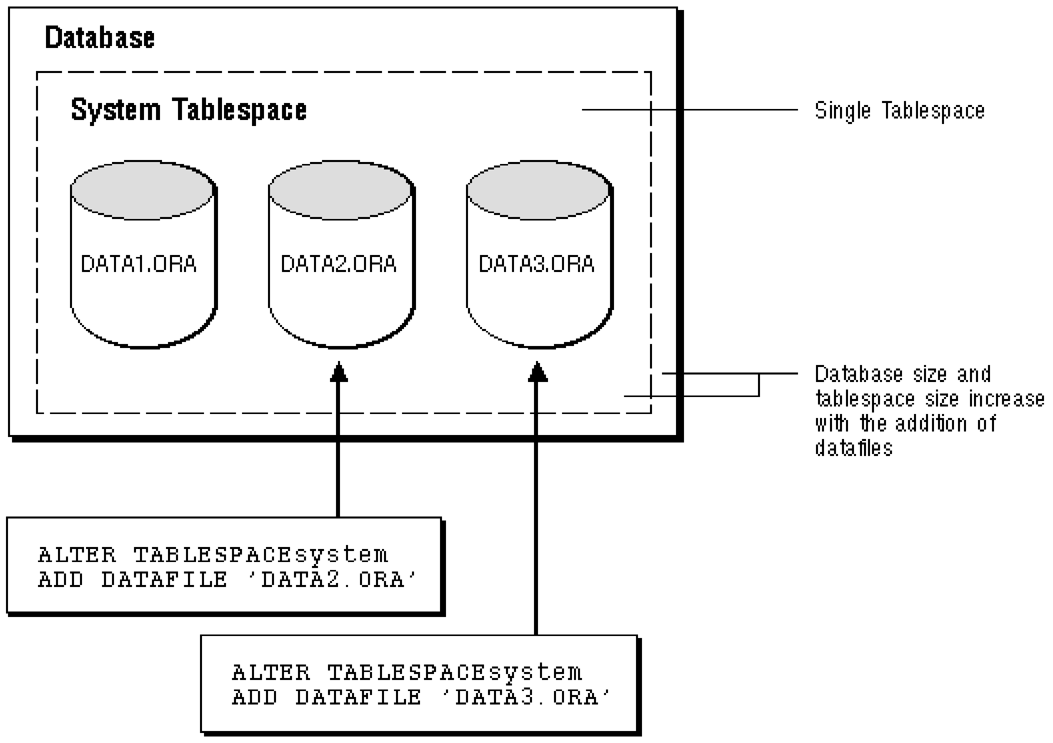
A small database might need only the SYSTEM tablespace; however, it is recommended that you create at least one additional tablespace to store user data separate from data dictionary information. This allows you more flexibility in various database administration operations and can reduce contention among dictionary objects and schema objects for the same datafiles.

***Note**: The SYSTEM tablespace must always be kept online.*

All data stored on behalf of stored PL/SQL program units (procedures, functions, packages and triggers) resides in the SYSTEM tablespace. If you create many of these PL/SQL objects, the database administrator needs to plan for the space in the SYSTEM tablespace that these objects use.

Allocating More Space for a Database

To enlarge a database, you have three options. You can add another datafile to one of its existing tablespaces, thereby increasing the amount of disk space allocated for the corresponding tablespace.

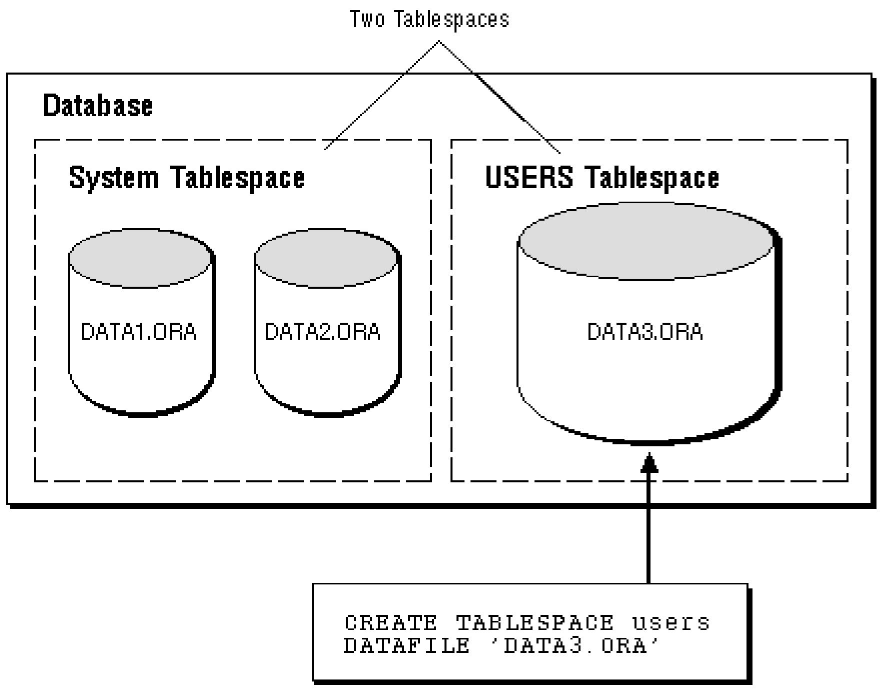

Database
System Tablespace
DATA1.ORA
DATA2.ORA
DATA3.ORA
Single Tablespace
Database size and tablespace size increase with the addition of datafiles

```
ALTER TABLESPACEsystem
ADD DATAFILE 'DATA2.ORA'
```

```
ALTER TABLESPACEsystem
ADD DATAFILE 'DATA3.ORA'
```

Figure: Enlarging a Database by Adding a Datafile to a Tablespace

Alternatively, a database administrator can create a new tablespace (defined by an additional datafile) to increase the size of a database.



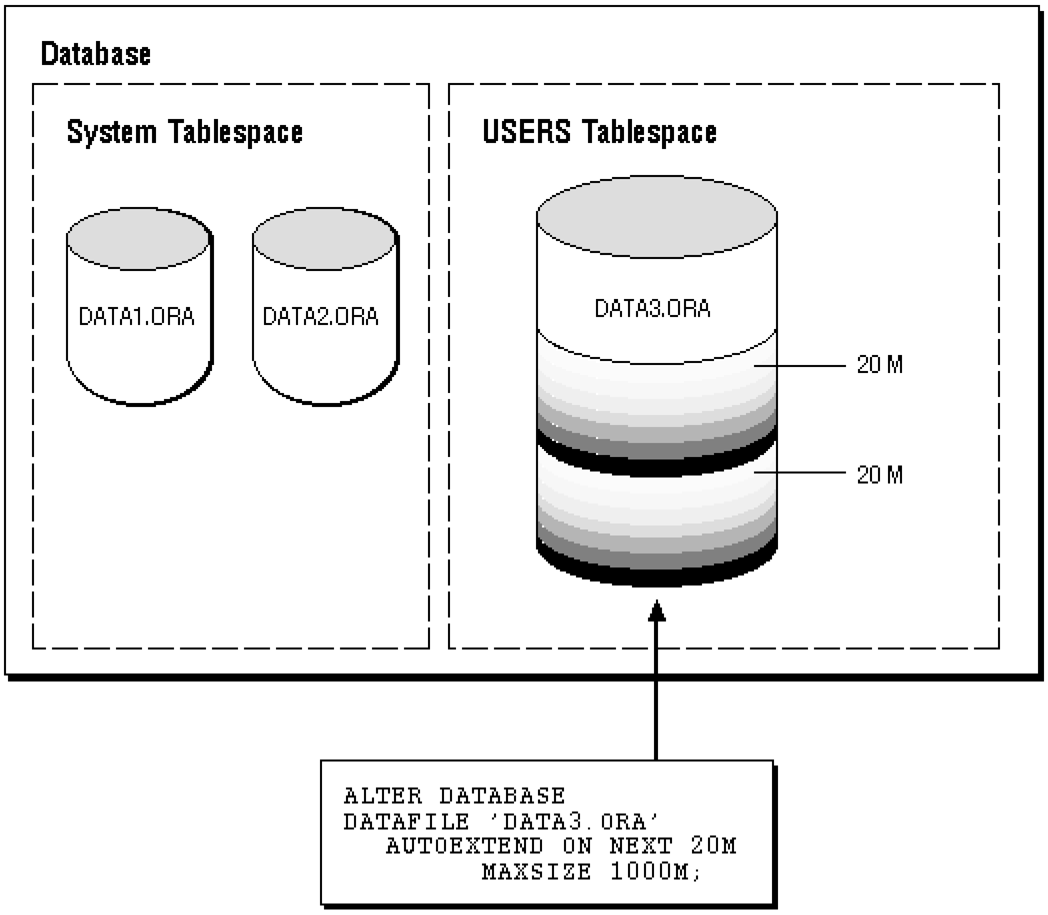
```
CREATE TABLESPACE users
DATAFILE 'DATA3.ORA'
```

Figure: Enlarging a Database by Adding a New Tablespace

www.csitportal.com

The size of a tablespace is the size of the datafile(s) that constitute the tablespace, and the size of a database is the collective size of the tablespaces that constitute the database.

The third option is to change a datafile's size or allow datafiles in existing tablespaces to grow dynamically as more space is needed. You accomplish this by altering existing files or by adding files with dynamic extension properties.



```
ALTER DATABASE
DATAFILE 'DATA3.ORA'
   AUTOEXTEND ON NEXT 20M
          MAXSIZE 1000M;
```

Figure: Enlarging a Database by Dynamically Sizing Datafiles

Online and Offline Tablespaces
A database administrator can bring any tablespace (except the SYSTEM tablespace) in an Oracle database *online* (accessible) or *offline* (not accessible) whenever the database is open.

Note: The SYSTEM tablespace must always be online because the data dictionary must always be available to Oracle.

A tablespace is normally online so that the data contained within it is available to database users. However, the database administrator might take a tablespace offline for any of the following reasons:

- to make a portion of the database unavailable, while allowing normal access to the remainder of the database
- to perform an offline tablespace backup (although a tablespace can be backed up while online and in use)
- to make an application and its group of tables temporarily unavailable while updating or maintaining the application

When a Tablespace Goes Offline

When a tablespace goes offline, Oracle does not permit any subsequent SQL statements to reference objects contained in the tablespace. Active transactions with completed statements that refer to data in a tablespace that has been taken offline are not affected at the transaction level. Oracle saves rollback data corresponding to statements that affect data in the offline tablespace in a deferred rollback segment (in the SYSTEM tablespace). When the tablespace is brought back online, Oracle applies the rollback data to the tablespace, if needed.

Note: You cannot take a tablespace offline if it contains any rollback segments that are in use.

When a tablespace goes offline or comes back online, it is recorded in the data dictionary in the SYSTEM tablespace. If a tablespace was offline when you shut down a database, the tablespace remains offline when the database is subsequently mounted and reopened.

You can bring a tablespace online only in the database in which it was created because the necessary data dictionary information is maintained in the SYSTEM tablespace of that database. An offline tablespace cannot be read or edited by any utility other than Oracle. Thus, tablespaces cannot be transferred from database to database.

Oracle automatically changes a tablespace from online to offline when certain errors are encountered (for example, when the database writer process, DBWR, fails in several attempts to write to a datafile of the tablespace). Users trying to access tables in the tablespace with the problem receive an error. If the problem that causes this disk I/O to fail is media failure, the tablespace must be recovered after you correct the hardware problem.

Using Tablespaces for Special Procedures

By using multiple tablespaces to separate different types of data, the database administrator can also take specific tablespaces offline for certain procedures, while other tablespaces remain online and the information in them is still available for use. However, special circumstances can occur when tablespaces are taken offline. For example, if two tablespaces are used to separate table data from index data, the following is true:

- If the tablespace containing the indexes is offline, queries can still access table data because queries do not require an index to access the table data.

- If the tablespace containing the tables is offline, the table data in the database is not accessible because the tables are required to access the data.

In summary, if Oracle determines that it has enough information in the online tablespaces to execute a statement, it will do so. If it needs data in an offline tablespace, then it causes the statement to fail.

Read-Only Tablespaces
The primary purpose of read-only tablespaces is to eliminate the need to perform backup and recovery of large, static portions of a database. Oracle never updates the files of a read-only tablespace, and therefore the files can reside on read-only media, such as CD ROMs or WORM drives.

Note: Because you can only bring a tablespace online in the database in which it was created, read-only tablespaces are not meant to satisfy archiving or data publishing requirements.

Whenever you create a new tablespace, it is always created as read-write. The READ ONLY option of the ALTER TABLESPACE command allows you to change the tablespace to read-only, making all of its associated datafiles read-only as well. You can then use the READ WRITE option to make a read-only tablespace writeable again.

Read-only tablespaces cannot be modified. Therefore, they do not need repeated backup. Also, should you need to recover your database, you do not need to recover any read-only tablespaces, because they could not have been modified.

You can drop items, such as tables and indexes, from a read-only tablespace, just as you can drop items from an offline tablespace. However, you cannot create or alter objects in a read-only tablespace.

Making a Tablespace Read-Only
Use the SQL command ALTER TABLESPACE to change a tablespace to read-only.

Read-Only vs. Online or Offline
Making a tablespace read-only does not change its offline or online status.

Offline datafiles cannot be accessed. Bringing a datafile in a read-only tablespace online makes the file readable. The file cannot be written to unless it’s associated tablespace is returned to the read-write state. The files of a read-only tablespace can independently be taken online or offline using the DATAFILE option of the ALTER DATABASE command.

Restrictions on Read-Only Tablespaces
You cannot add datafiles to a tablespace that is read-only, even if you take the tablespace offline. When you add a datafile, Oracle must update the file header, and this write operation is not allowed.

To update a read-only tablespace, you must first make the tablespace writeable. After updating the tablespace, you can then reset it to be read-only.

Temporary Tablespaces
Space management for sort operations is performed more efficiently using temporary tablespaces designated exclusively for sorts. This scheme effectively eliminates serialization of space management operations involved in the allocation and de-allocation of sort space. All operations that use sorts, including joins, index builds, ordering (ORDER BY), the computation of aggregates (GROUP BY), and the ANALYZE command to collect optimizer statistics, benefit from temporary tablespaces. The performance gains are significant in parallel server environments.

A *temporary tablespace* is a tablespace that can only be used for sort segments. No permanent objects can reside in a temporary tablespace. Sort segments are used when a segment is shared by multiple sort operations. One sort segment exists in every instance that performs a sort operation in a given tablespace.

Temporary tablespaces provide performance improvements when you have multiple sorts that are too large to fit into memory. The sort segment of a given temporary tablespace is created at the time of the first sort operation. The sort segment grows by allocating extents until the segment size is equal to or greater than the total storage demands of all of the active sorts running on that instance.

You create temporary tablespaces using the following SQL syntax:

```
CREATE TABLESPACE tablespace TEMPORARY
```

You can also alter a tablespace from PERMANENT to TEMPORARY or vice versa using the following syntax:

```
ALTER TABLESPACE tablespace TEMPORARY
```

Datafiles
A tablespace in an Oracle database consists of one or more physical *datafiles*. A datafile can be associated with only one tablespace, and only one database.

When a datafile is created for a tablespace, Oracle creates the file by allocating the specified amount of disk space plus the overhead required for the file header. When a datafile is created, the operating system is responsible for clearing old information and authorizations from a file before allocating it to Oracle. If the file is large, this process might take a significant amount of time.

Additional Information: For information on the amount of space required for the file header of datafiles on your operating system, see your Oracle operating system specific documentation.

Since the first tablespace in any database is always the SYSTEM tablespace, Oracle automatically allocates the first datafiles of any database for the SYSTEM tablespace during database creation.

Datafile Contents

After a datafile is initially created, the allocated disk space does not contain any data; however, Oracle reserves the space to hold only the data for future segments of the associated tablespace -- it cannot store any other program's data. As a segment (such as the data segment for a table) is created and grows in a tablespace, Oracle uses the free space in the associated datafiles to allocate extents for the segment.

The data in the segments of objects (data segments, index segments, rollback segments, and so on) in a tablespace are physically stored in one or more of the datafiles that constitute the tablespace. Note that a schema object does not correspond to a specific datafile; rather, a datafile is a repository for the data of any object within a specific tablespace. Oracle allocates the extents of a single segment in one or more datafiles of a tablespace; therefore, an object can "span" one or more datafiles. Unless table "striping" is used, the database administrator and end-users cannot control which datafile stores an object.

Size of Datafiles

You can alter the size of a datafile after its creation or you can specify that a datafile should dynamically grow as objects in the tablespace grow. This functionality allows you to have fewer datafiles per tablespace and can simplify administration of datafiles.

Offline Datafiles

You can take tablespaces *offline* (make unavailable) or bring them *online* (make available) at any time. Therefore, all datafiles making up a tablespace are taken offline or brought online as a unit when you take the tablespace offline or bring it online, respectively. You can take individual datafiles offline; however, this is normally done only during certain database recovery procedures.

Control and Redo Log Files

Maintaining the Control File

You can think of the **control file** as a metadata repository for the physical database. It has the structure of the database—the data files and redo log files that constitute a database. The control file is a binary file, created when the database is created, and is updated with the physical changes whenever you add or rename a file.

The control file is updated continuously and should be available at all times. Don’t edit the contents of the control file; only Oracle processes should update its contents. When you start up the database, Oracle uses the control file to identify the data files, redo log files, and open them. Control files play a major role when recovering a database.

The contents of the control file include the following:

- ✓ Database name to which the control file belongs. A control file can belong to only one database.
- ✓ Database creation timestamp.
- ✓ Data files—name, location, and online/offline status information.
- ✓ Redo log files—name and location.
- ✓ Redo log archive information.
- ✓ Tablespace names.
- ✓ Current *log sequence number*, a unique identifier that is incremented and recorded when an online redo log file is switched.
- ✓ Most recent checkpoint information. A *checkpoint* occurs when all the modified database buffers in the SGA are written to the data files. The *system change number* (SCN), a number sequentially assigned to each transaction in the database, is also recorded in the control file against the data file name that is taken offline or made read-only.
- ✓ Begin and end of undo segments.
- ✓ Recovery Manager's (RMAN's) backup information. RMAN is the Oracle utility you use to back up and recover databases.

The control file size is determined by the MAX clauses you provide when you create the database: MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, MAXDATAFILES, and MAXINSTANCES. Oracle pre-allocates space for these maximums in the control file. Therefore, when you add or rename a file in the database, the control file size does not change.

When you add a new file to the database or relocate a file, an Oracle server process immediately updates the information in the control file. Back up the control file after any structural changes. The log writer process (LGWR) updates the control file with the current log sequence number. The checkpoint process (CKPT) updates the control file with the recent checkpoint information. When the database is in ARCHIVELOG mode, the archiver process (ARCn) updates the control file with archiving information such as the archive log file name and log sequence number.

The control file contains two types of *record sections*: reusable and not reusable. Recovery Manager Information is kept in the reusable section. Items such as the names of the backup data files are kept in this section, and once this section fills up, the entries are re-used in a circular fashion.

Multiplexing Control Files

Since the control file is critical for the database operation, Oracle recommends a minimum of two control files. You duplicate the control file on different disks either by using the multiplexing feature of Oracle or by using the mirroring feature of your operating system. The next two sections discuss the two ways you can implement the multiplexing feature: using *init.ora* and using an SPFILE.

Multiplexing Control Files Using init.ora (Pfile)

Multiplexing is defined as keeping a copy of the same control file in different locations. Copying the control file to multiple locations and changing the CONTROL_FILES parameter in the initialization file *init.ora* to include all control file names specifies the multiplexing of the control file. The following syntax shows three multiplexed control files.

```
CONTROL_FILES = ('/ora01/oradata/MYDB/ctrlMYDB01.ctl',
'/ora02/oradata/MYDB/ctrlMYDB02.ctl', '/ora03/oradata/MYDB/ctrlMYDB03.ctl')
```

By storing the control file on multiple disks, you avoid the risk of a single point of failure. When multiplexing control files, updates to the control file can take a little longer, but that is insignificant when compared with the benefits. If you lose one control file, you can restart the database after copying one of the other control files or after changing the CONTROL_FILES parameter in the initialization file.

When multiplexing control files, Oracle updates all the control files at the same time, but uses only the first control file listed in the CONTROL_FILES parameter for reading.

When creating a database, you can list the control file names in the CONTROL_FILES parameter, and Oracle creates as many control files as are listed. You can have a maximum of eight multiplexed control file copies.

If you need to add more control file copies, do the following:

1. Change the initialization parameter file to include the new control file name(s) in the parameter CONTROL_FILES.
2. Shut down the database.
3. Copy the control file to more locations by using an operating system command.
4. Start up the database.

After creating the database, you can change the location of the control files, rename the control files, or drop certain control files. You must have at least one control file for each database. To add, rename, or delete control files, you need to follow the preceding steps. Basically, you shut down the database, use the operating system copy command (copy, rename, or drop the control files accordingly), edit the CONTROL_FILES parameter in *init.ora* and start up the database.

Multiplexing Control Files Using an SPFILE

Multiplexing using an SPFILE is similar to multiplexing using *init.ora*; the major difference being how the CONTROL_FILES parameter is changed.

Follow these steps:

1. Alter the SPFILE while the database is still open:

   ```
   SQL> ALTER SYSTEM SET CONTROL_FILES =

   '/ora01/oradata/MYDB/ctrlMYDB01.ctl', '/ora02/oradata/MYDB/ctrlMYDB02.ctl',
   '/ora03/oradata/MYDB/ctrlMYDB03.ctl', '/ora04/oradata/MYDB/ctrlMYDB04.ctl'
   SCOPE=SPFILE;
   ```

   This parameter change will only take effect after the next instance restart by using the SCOPE=SPFILE qualifier. The contents of the binary SPFILE are changed immediately, but the old specification of CONTROL_FILES will be used until the instance is restarted.

2. Shut down the database.

   ```
   SQL> SHUTDOWN NORMAL
   ```

3. Copy an existing control file to the new location:

   ```
   $cp /ora01/oradata/MYDB/ctrlMYDB01.ctl

   /ora01/oradata/MYDB/ctrlMYDB04.ctl
   ```

4. Start the instance.

   ```
   SQL> STARTUP
   ```

TIP: If you lose one of the control files, you can shut down the database, copy a control file, or change the CONTROL_FILES parameter and restart the database.

Creating New Control Files

You can create a new control file by using the CREATE CONTROLFILE command. You will need to create a new control file if you lose all the control files that belong to the database, if you want to change any of the MAX clauses in the CREATE DATABASE command, or if you want to change the database name. You must know the data file names and redo log file names to create the control file. Follow these steps to create the new control file:

1. Prepare the CREATE CONTROLFILE command. You should have the complete list of data files and redo log files. If you omit any data files, they can no longer be a part of the database. The following is an example of the CREATE CONTROLFILE command.

```
CREATE CONTROLFILE SET DATABASE "ORACLE"  NORESETLOGS
NOARCHIVELOG  MAXLOGFILES 32

MAXLOGMEMBERS 2

MAXDATAFILES 32

MAXINSTANCES 1

        MAXLOGHISTORY 1630  LOGFILE

GROUP 1 ‘C:\ORACLE\DATABASE\LOG2ORCL.ORA’ SIZE 500K,  GROUP 2
‘C:\ORACLE\DATABASE\LOG1ORCL.ORA’ SIZE 500K

DATAFILE

‘C:\ORACLE\DATABASE\SYS1ORCL.ORA’,
‘C:\ORACLE\DATABASE\USR1ORCL.ORA’,
‘C:\ORACLE\DATABASE\RBS1ORCL.ORA’,
‘C:\ORACLE\DATABASE\TMP1ORCL.ORA’,
‘C:\ORACLE\DATABASE\APPDATA1.ORA’,

‘C:\ORACLE\DATABASE\APPINDX1.ORA’;
```

The options in this command are similar to the CREATE DATABASE command. The NORESETLOGS option specifies that the online redo log files should not be reset.

2. Shut down the database.
3. Start up the database with the NOMOUNT option. Remember, to mount the database, Oracle needs to open the control file.
4. Create the new control file with a command similar to the preceding example. The control files will be created using the names and locations specified in the initialization parameter CONTROL_FILES.
5. Open the database by using the ALTER DATABASE OPEN command.
6. Shut down the database and back up the database.

These steps are very basic. Depending on the situation, you might have to perform additional steps.

After creating the control file, determine whether any of the data files listed in the dictionary are missing in the control file. If you query the V$DATAFILE view, the missing files will have the name MISSING. If you created the control file by using the RESETLOGS option, the missing data files cannot be added back to the database. If you created the control file with the NORESETLOGS option, the missing data file can be included in the database by performing a media recovery.

You can back up the control file when the database is up by using the command

```
ALTER DATABASE BACKUP CONTROLFILE TO '<filename>' REUSE;
```

Querying Control File Information

The Oracle data dictionary holds all the information about the control file. The view V$CONTROLFILE lists the names of the control files for the database. The STATUS column should always be NULL; when a control file is missing, the STATUS would be INVALID, but that should never occur because when Oracle cannot update one of the control files, the instance crashes – you can start up the database only after copying a good control file.

For example, in order to obtain control information you do as following:

```
SQL> SELECT * FROM V$CONTROLFILE;

STATUS NAME

------ -------------------------------------

/ora01/oradata/MYDB/ctrlMYDB01.ctl

/ora02/oradata/MYDB/ctrlMYDB02.ctl

/ora03/oradata/MYDB/ctrlMYDB03.ctl  3 rows selected.

SQL>
```

You can also use the SHOW PARAMETER command to retrieve the names of the control files.

```
SQL> show parameter control_files

NAME TYPE VALUE

---------------- ----------- ------------------------------

control_files string H:\Oracle\oradata\or90\CONTROL01.CTL,

H:\Oracle\oradata\or90\CONTROL02.CTL,
```

H:\Oracle\oradata\or90\CONTROL03.CTL

Maintaining and Monitoring Redo Log Files

Redo logs record all changes to the database. The redo log buffer in the SGA is written to the redo log file periodically by the LGWR process. The redo log files are accessed and are open during normal database operation; hence they are called the online redo log files. Every Oracle database must have at least two redo log files. The LGWR process writes to these files in a circular fashion. For example, say there are three online redo log files. The LGWR process writes to the first file, and when this file is full, it starts writing to the second file, and then to the third file, and then again to the first file (overwriting the contents).

Online redo log files are filled with redo records. A *redo record*, also called a redo entry, is made up of a group of *change vectors*, each of which is a description of a change made to a single block in the database. Redo entries record data that you can use to reconstruct all changes made to the database, including the undo segments. When you recover the database by using redo log files, Oracle reads the change vectors in the redo records and applies the changes to the relevant blocks.

LGWR writes redo information from the redo log buffer to the online redo log files under a variety of circumstances: A user commits a transaction, even if this is the only transaction in the log buffer.

- ✓ The redo log buffer becomes one-third full.
- ✓ When there is approximately 1MB of changed records in the buffer. This total does not include deleted or inserted records.

NOTE: LGWR always writes its records to the online redo log file *before* DBW*n* writes new or modified database buffer cache records to the datafiles.

Each database has its own online *redo log groups*. A log group can have one or more *redo log members* (each member is a single operating system file). If you have a Real Application Cluster configuration, in which multiple instances are mounted to one database, each instance will have one online redo thread. That is, the LGWR process of each instance writes to the same online redo log files, and hence Oracle has to keep track of the instance from where the database changes are coming. For single instance configurations, there will be only one thread, and that thread number is 1. The redo log file contains both committed and uncommitted transactions. Whenever a transaction is committed, a system change number is assigned to the redo records to identify the committed transaction.

The redo log group is referenced by an integer; you can specify the group number when you create the redo log files, either when you create the database or when you create the control file. You can also change the redo log configuration (add/drop/rename files) by using database commands. The following example shows a CREATE DATABASE command.

```
CREATE DATABASE “MYDB01”

LOGFILE ‘/ora02/oradata/MYDB01/redo01.log’ SIZE 10M,
‘/ora03/oradata/MYDB01/redo02.log’ SIZE 10M;
```

Two log file groups are created here; the first file will be assigned to group 1, and the second file will be assigned to group 2. You can have more files in each group; this practice is known as the multiplexing of redo log files, which we’ll discuss later. You can specify any group number — the range will be between 1 and MAXLOGFILES. Oracle recommends that all redo log groups be the same size. The following is an example of creating the log files by specifying the groups.

```
CREATE DATABASE “MYDB01”

LOGFILE GROUP 1 ‘/ora02/oradata/MYDB01/redo01.log’ SIZE 10M,  GROUP 2
‘/ora03/oradata/MYDB01/redo02.log’ SIZE 10M;
```

Log Switch Operations

The LGWR process writes to only one redo log file group at any time. The file that is actively being written to is known as the current log file. The log files that are required for instance recovery are known as the active log files. The other log files are known as inactive. Oracle automatically recovers an instance when starting up the instance by using the online redo log files. Instance recovery may be needed if you do not shut down the database properly or if your computer crashes.

The log files are written in a circular fashion. A log switch occurs when Oracle finishes writing to one file and starts writing to the next file. A log switch always occurs when the current redo log file is completely full and log writing must continue. You can force a log switch by using the ALTER SYSTEM command. A manual log switch may be necessary when performing maintenance on the redo log files by using the ALTER SYSTEM SWITCH LOGFILE command. Figure 1 shows how LGWR writes to the redo log groups in a circular fashion.

Whenever a log switch occurs, Oracle allocates a sequence number to the new redo log file before writing to it. As stated earlier, this number is known as the log sequence number. If there are lots of transactions or changes to the database, the log switches can occur too frequently. Size the redo log file appropriately to avoid frequent log switches. Oracle writes to the alert log file whenever a log switch occurs.

www.csitportal.com

www.csitportal.com

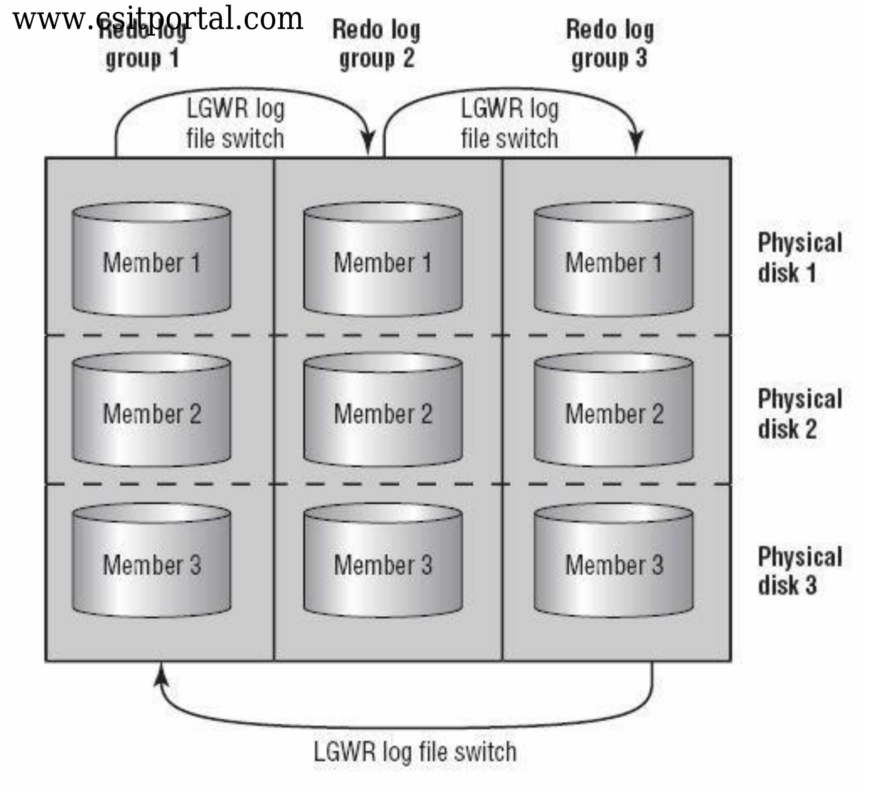


Fig. 1. Redo log file usage.

Database checkpoints are closely tied to redo log file switches. A checkpoint is an event that flushes the modified data from the buffer cache to the disk and updates the control file and data files. The CKPT process updates the headers of data files and control files; the actual blocks are written to the file by the DBW*n* process. A checkpoint is initiated when the redo log file is filled and a log switch occurs, when the instance is shut down with NORMAL, TRANSACTIONAL, or IMMEDIATE, when a tablespace status is changed to readonly or put into BACKUP mode, when a tablespace or datafile is taken offline, or when other values specified by certain parameters (discussed later) are reached.

You can force a checkpoint if needed. Forcing a checkpoint ensures that all changes to the database buffers are written to the data files on disk.

```
ALTER SYSTEM CHECKPOINT;
```

Another way to force a checkpoint is by forcing a log file switch.

```
ALTER SYSTEM SWITCH LOGFILE;
```

Multiplexing Log Files

You can keep multiple copies of the online redo log file to safeguard against damage to these files. When multiplexing online redo log files, LGWR concurrently writes the same redo log information to multiple identical online redo log files, thereby eliminating a single point of redo log failure. All copies of the redo file are the same size and are known as a *group*, which is identified by an integer. Each redo log file in the group is known as a *member*. You must have at least two redo log groups for normal database operation.
When multiplexing redo log files, it is preferable to keep the members of a group on different disks, so that one disk failure will not affect the continuing operation of the database. If LGWR can write to at least one member of the group, database operation proceeds as normal; an entry is written to the alert log file. If all members of the redo log file group are not available for writing, Oracle shuts down the instance. An instance recovery or media recovery may be needed to bring up the database.

You can create multiple copies of the online redo log files when you create the database. For example, the following statement creates two redo log file groups with two members in each.

CREATE DATABASE “MYDB01” LOGFILE

GROUP 1 (‘/ora02/oradata/MYDB01/redo0101.log’, ‘/ora03/oradata/MYDB01/redo0102.log’)

SIZE 10M,

GROUP 2 (‘/ora02/oradata/MYDB01/redo0201.log’, ‘/ora03/oradata/MYDB01/redo0202.log’)

SIZE 10M;

The maximum number of log file groups is specified in the clause MAXLOGFILES, and the maximum number of members is specified in the clause MAXLOGMEMBERS. You can separate the filenames (members) by using a space or a comma.

Creating New Groups

You can create and add more redo log groups to the database by using the ALTER DATABASE command. The following statement creates a new log file group with two members.

ALTER DATABASE ADD LOGFILE

GROUP 3 (‘/ora02/oradata/MYDB01/redo0301.log’, ‘/ora03/oradata/MYDB01/redo0402.log’)

SIZE 10M;

If you omit the GROUP clause, Oracle assigns the next available number. For example, the following statement also creates a multiplexed group.

ALTER DATABASE ADD LOGFILE

('/ora02/oradata/MYDB01/redo0301.log' '/ora03/oradata/MYDB01/redo0402.log') SIZE 10M;

To create a new group without multiplexing, use the following statement.

ALTER DATABASE ADD LOGFILE

'/ora02/oradata/MYDB01/redo0301.log' REUSE;

You can add more than one redo log group by using the ALTER DATABASE command—just use a comma to separate the groups.

TIP: If the redo log files you create already exist, use the REUSE option and don't specify the size. The new redo log size will be same as that of the existing file.

Adding New Members

If you forgot to multiplex the redo log files when creating the database or if you need to add more redo log members, you can do so by using the ALTER DATABASE command. When adding new members, you do not specify the file size, because all group members will have the same size.

If you know the group number, using the following statement will add a member to group 2.

ALTER DATABASE ADD LOGFILE MEMBER

'/ora04/oradata/MYDB01/redo0203.log' TO GROUP 2;

You can also add group members by specifying the names of other members in the group, instead of specifying the group number. Specify all the existing group members with this syntax.

ALTER DATABASE ADD LOGFILE MEMBER

'/ora04/oradata/MYDB01/redo0203.log' TO ('/ora02/oradata/MYDB01/redo0201.log',

'/ora03/oradata/MYDB01/redo0202.log');

Renaming Log Members

If you want to move the log file member from one disk to another or just want a more meaningful name, you can rename a redo log member. Before renaming the online redo log members, the new (target) online redo files should exist. The SQL commands in Oracle change only the internal pointer in the control file to a new log file; they do not change or rename the operating system file. You must use an operating system command to rename or move the file. Follow these steps to rename a log member:

1. Shut down the database (a complete backup is recommended).
2. Copy/rename the redo log file member to the new location by using an operating system command.
3. Start up the instance and mount the database (STARTUP MOUNT).
4. Rename the log file member in the control file. Use ALTER DATABASE RENAME FILE '<old_redo_file_name>' TO '<new_redo_file_name>';
5. Open the database (ALTER DATABASE OPEN).
6. Back up the control file.

Dropping Redo Log Groups

You can drop a redo log group and its members by using the ALTER DATABASE command. Remember that you should have at least two redo log groups for the database to function normally. The group that is to be dropped should not be the active group or the current group—that is, you can drop only an inactive log file group. If the log file to be dropped is not inactive, use the ALTER SYSTEM SWITCH LOGFILE command.

To drop the log file group 3, use the following SQL statement.

ALTER DATABASE DROP LOGFILE GROUP 3;

When an online redo log group is dropped from the database, the operating system files are not deleted from disk. The control files of the associated database are updated to drop the members of the group from the database structure. After dropping an online redo log group, make sure that the drop is completed successfully, and then use the appropriate operating system command to delete the dropped online redo log files.

Dropping Redo Log Members

Similar to conditions for dropping a redo log group, you can drop only the members of an inactive redo log group. Also, if there are only two groups, the log member to be dropped should not be the last member of a group. You can have a different number of members for each redo log group, though it is not advised. For example, say you have three log groups, each with two members. If you drop a log member from group 2, and a failure occurs to the sole member of group 2, the instance crashes. So even if you drop a member for maintenance reasons, ensure that all redo log groups have the same number of members.

To drop the log member, use the DROP LOGFILE MEMBER clause of the ALTER DATABASE command.

ALTER DATABASE DROP LOGFILE MEMBER

'/ora04/oradata/MYDB01/redo0203.log';

The operating system file is not removed from the disk; only the control file is updated. Use an operating system command to delete the redo log file member from disk.

TIP: If a database is running in ARCHIVELOG mode, redo log members cannot be deleted unless the redo log group has been archived.

Clearing Online Redo Log Files

Under certain circumstances, a redo log group member (or all members of a log group) may become corrupted. To solve this problem, you can drop and re-add the log file group or group member. It is much easier, however, to use the ALTER DATABASE CLEAR LOGFILE command. The following example clears the contents of redo log group 3 in the database:

ALTER DATABASE CLEAR LOGFILE GROUP 3;

Another distinct advantage of this command is that you can clear a log group even if the database has only two log groups, and only one member in each group. You can also clear a log group member even if it has not been archived yet by using the UNARCHIVED keyword. In this case, it is advisable to do a full database backup at the earliest convenience, because the unarchived redo log file is no longer usable for database recovery.

Setting ARCHIVELOG

Specifying these parameters does not start writing the archive log files; you should place the database in ARCHIVELOG mode to enable archiving of the redo log files. You can specify the ARCHIVELOG clause while creating the database. However, most DBAs prefer to create the database first and then enable ARCHIVELOG mode. To enable ARCHIVELOG mode, follow these steps:

1. Shut down the database. Set up the appropriate initialization parameters.
2. Start up and mount the database.
3. Enable ARCHIVELOG mode by using the command ALTER DATABASE ARCHIVELOG.
4. Open the database by using ALTER DATABASE OPEN.

To disable ARCHIVELOG mode, follow these steps:

1. Shut down the database.

www.csitportal.com

2. Start up and mount the database.
3. Disable ARCHIVELOG mode by using the command ALTER DATABASE NOARCHIVELOG.
4. Open the database by using ALTER DATABASE OPEN.

You can enable automatic archiving by setting the parameter LOG_ARCHIVE_START = TRUE. If you set the parameter to FALSE, Oracle does not start the ARC*n* process. Therefore, when the redo log files are full, the database will hang, waiting for the redo log files to be archived. You can initiate the automatic archive process by using the command ALTER SYSTEM ARCHIVE LOG START, which starts the ARC*n* processes; to manually archive all unarchived logs, use the command

ALTER SYSTEM ARCHIVE LOG ALL.

Querying Log and Archive Information

You can query the redo log file information from the SQL command ARCHIVE LOG LIST or by querying the dynamic performance views.

The ARCHIVE LOG LIST command shows whether the database is in ARCHIVELOG mode, whether automatic archiving is enabled, the archival destination, and the oldest, next, and current log sequence numbers.

```
SQL> archive log list

Database log mode              Archive Mode

Automatic archival             Enabled

Archive destination            C:\Oracle\oradata\ORADB02\archive

Oldest online log sequence    194

Next log sequence to archive  196

Current log sequence          196

SQL>
```

TIP: The view V$DATABASE shows whether the database is in ARCHIVELOG mode or in NOARCHIVELOG mode.

V$LOG

This dynamic performance view contains information about the log file groups and sizes and its status. The valid status codes in this view and their meanings are as follows:

UNUSED New log group, never used.

CURRENT Current log group.

ACTIVE Log group that may be required for instance recovery.

CLEARING You issued an ALTER DATABASE CLEAR LOGFILE command.

CLEARING_CURRENT Empty log file after issuing the ALTER DATABASE CLEAR LOGFILE command.

INACTIVE The log group is not needed for instance recovery.

V$LOGFILE

The V$LOGFILE view has information about the log group members. The filenames and group numbers are in this view. The STATUS column can have the value INVALID (file is not accessible), STALE (file's contents are incomplete), DELETED (file is no longer used), or blank (file is in use).

```
SQL> SELECT * FROM V$LOGFILE

2                      ORDER BY GROUP#;  GROUP# STATUS TYPE MEMBER

-------- ------- ------- -----------------------------------


          1            ONLINE    C:\ORACLE\ORADATA\ORADB02\REDO11.L
          1            ONLINE    D:\ORACLE\ORADATA\ORADB02\REDO12.L
          2  STALE     ONLINE    C:\ORACLE\ORADATA\ORADB02\REDO21.L
          2            ONLINE    D:\ORACLE\ORADATA\ORADB02\REDO22.LO
          3            ONLINE    C:\ORACLE\ORADATA\ORADB02\REDO31.LO
          3            ONLINE    D:\ORACLE\ORADATA\ORADB02\REDO32.LO
6 rows selected.
```

Managing Users and Security

Profiles

You use profiles to control the database and system resource usage. Oracle provides a set of predefined resource parameters that you can use to monitor and control database resource usage. You can define limits for each resource by using a database profile. You also use profiles for password management. You can create profiles for different user communities and then assign a profile to each user. When you create the database, Oracle creates a profile named DEFAULT, and if you do not specify a profile for the user, Oracle assigns the user the DEFAULT profile.

Managing Resources

Oracle lets you control the following types of resource usage through profiles:

- Concurrent sessions per user
- Elapsed and idle time connected to the database
- CPU time used
- Private SQL and PL/SQL area used in the SGA (System Global Area)
- Logical reads performed
- Amount of private SGA space used in Shared Server configurations

Resource limits are enforced only if you have set the parameter RESOURCE_LIMIT to TRUE. Even if you have defined profiles and assigned profiles to users, Oracle enforces them only when this parameter is set to TRUE. You can set this parameter in the initialization parameter file so that every time the database starts, the resource usage is controlled for each user using the assigned profile. You enable or disable resource limits using the ALTER SYSTEM command. The default value of RESOURCE_LIMIT is FALSE.

The limits for each resource are specified as an integer; you can set no limit for a given resource by specifying UNLIMITED, or you can use the value specified in the DEFAULT profile by specifying DEFAULT. The DEFAULT profile initially has the value UNLIMITED for all resources. After you create the database, you can modify the DEFAULT profile.

Most resource limits are set at the session level; a session is created when a user connects to the database. You can control certain limits at the statement level (but not at the transaction level). If a user exceeds a resource limit, Oracle aborts the current operation, rolls back the changes made by the statement, and returns an error. The user has the option of committing or rolling back the transaction, because the statements issued earlier in the transaction are not aborted. No other operation is permitted when a session-level limit is reached. The user can disconnect, in which case the transaction is committed. You use the following parameters to control resources:

SESSIONS_PER_USER Limits the number of concurrent user sessions. No more sessions from the current user are allowed when this threshold is reached.

CPU_PER_SESSION Limits the amount of CPU time a session can use. The CPU time is specified in hundredths of a second.

CPU_PER_CALL Limits the amount of CPU time a single SQL statement can use. The CPU time is specified in hundredths of a second. This parameter is useful for controlling runaway queries, but you should be careful to specify this limit for batch programs.

LOGICAL_READS_PER_SESSION Limits the number of data blocks read in a session, including the blocks read from memory and from physical reads.

LOGICAL_READS_PER_CALL Limits the number of data blocks read by a single SQL statement, including the blocks read from memory and from physical reads.

PRIVATE_SGA Limits the amount of space allocated in the SGA for private areas, per session. Private areas for SQL and PL/SQL statements are created in the SGA in the multithreaded architecture. You can specify K to indicate the size in KB or M to indicate the size in MB. If you specify neither K or M, the size is in bytes. This limit does not apply to dedicated server architecture connections.

CONNECT_TIME Specifies the maximum number of minutes a session can stay connected to the database (total elapsed time, not CPU time). When the threshold is reached, the user is automatically disconnected from the database; any pending transaction is rolled back.

IDLE_TIME Specifies the maximum number of minutes a session can be continuously idle, that is, without any activity for a continuous period of time. When the threshold is reached, the user is disconnected from the database; any pending transaction is rolled back.

COMPOSITE_LIMIT A weighted sum of four resource limits: CPU_PER_SESSION, LOGICAL_READS_PER_SESSION, CONNECT_TIME, and
PRIVATE_SGA. You can define a cost for the system resources (the resource cost on one database may be different from another, based on the number of transactions, CPU, memory, and so on) known as the composite limit. The upcoming section “Managing Profiles” discusses setting the resource cost.

Managing Password

You also use profiles to manage passwords. You can set the following by using profiles:

Account locking Number of failed login attempts, and the number of days the password will be locked.

Password expiration How often passwords should be changed, whether passwords can be reused, and the grace period after which the user is warned that the password change is due.

Password complexity Use of a customized function to verify the password complexity—for example, the password should not be the same as the user ID, cannot include commonly used words, and so on.

You can use the following parameters in profiles to manage passwords:

www.csitportal.com

FAILED_LOGIN_ATTEMPTS Specifies the maximum number of consecutive invalid login attempts (providing an incorrect password) allowed before the user account is locked.

PASSWORD_LOCK_TIME Specifies the number of days the user account will remain locked after the user has made FAILED_LOGIN_ATTEMPTS number of consecutive failed login attempts.

PASSWORD_LIFE_TIME Specifies the number of days a user can use one password. If the user does not change the password within the number of days specified, all connection requests return an error. The DBA then has to reset the password.

PASSWORD_GRACE_TIME Specifies the number of days the user will get a warning before the password expires. This is a reminder for the user to change the password.

PASSWORD_REUSE_TIME Specifies the number of days a password cannot be used again after changing it.

PASSWORD_REUSE_MAX Specifies the number of password changes required before a password can be reused. You cannot specify a value for both PASSWORD_REUSE_TIME and PASSWORD_REUSE_MAX; one should always be set to UNLIMITED, because you can enable only one type of password history method.

PASSWORD_VERIFY_FUNCTION Specifies the function you want to use to verify the complexity of the new password. Oracle provides a default script, which you can modify.

TIP: You can specify minutes or hours as a fraction or expression in parameters that require days as a value. One hour can be represented as 0.042 days or 1/24, and one minute can be specified as 0.000694 days, 1/24/60, or 1/1440.

Managing Profiles

You can create many profiles in the database that specify both resource management parameters and password management parameters. However, you can assign a user only one profile at any given time. To create a profile, you use the CREATE PROFILE command. You need to provide a name for the profile and then specify the parameter names and their values separated by space(s).

As an example, let’s create a profile to manage passwords and resources for the accounting department users. The users are required to change their password every 60 days, and they cannot reuse a password for 90 days. They are allowed to make a typo in the password only six consecutive times while connecting to the database; if the login fails a seventh time, their account is locked forever (until the DBA or security department unlocks the account).

The accounting department users are allowed a maximum of six database connections; they can stay connected to the database for 24 hours, but an inactivity of 2 hours will terminate their session. To prevent users from performing runaway queries, in this example we will set the maximum number of blocks they can read per SQL statement to 1 million.

```
SQL> CREATE PROFILE ACCOUNTING_USER LIMIT

SESSIONS_PER_USER 6

CONNECT_TIME 1440

IDLE_TIME 120

LOGICAL_READS_PER_CALL 1000000

PASSWORD_LIFE_TIME 60

PASSWORD_REUSE_TIME 90

PASSWORD_REUSE_MAX UNLIMITED

FAILED_LOGIN_ATTEMPTS 6

PASSWORD_LOCK_TIME UNLIMITED;
```

In the example, parameters such as PASSWORD_GRACE_TIME, CPU_PER_SESSION, and PRIVATE_SGA are not used. They will have a value of DEFAULT, which means the value will be taken from the DEFAULT profile.

The DBA or security officer can unlock a locked user account by using the ALTER USER command. The following example shows the unlocking of SCOTT's account.

```
SQL> ALTER USER SCOTT ACCOUNT UNLOCK;

User altered.
```

Composite Limit

The composite limit specifies the total resource cost for a session. You can define a weight for each resource based on the available resources. The following resources are considered for calculating the composite limit:

CPU_PER_SESSION

LOGICAL_READS_PER_SESSION

CONNECT_TIME

PRIVATE_SGA

The costs associated with each of these resources are set at the database level by using the

www.csitportal.com

ALTER RESOURCE COST command. By default, the resources have a cost of 0, which means they should not be considered for a composite limit (they are inexpensive). A higher cost means that the resource is expensive. If you do not specify any of these resources in ALTER RESOURCE COST, Oracle will keep the previous value. For example:

```
SQL> ALTER RESOURCE COST

  LOGICAL_READS_PER_SESSION 10

  CONNECT_TIME 2; Resource cost altered.
```

Here CPU_PER_SESSION and PRIVATE_SGA will have a cost of 0 (if they have not been modified before).

You can define limits for each of the four parameters in the profile as well as set the composite limit. The limit that is reached first is the one that takes effect. The following statement adds a composite limit to the profile you created earlier.

```
SQL> ALTER PROFILE ACCOUNTING_USER LIMIT

COMPOSITE_LIMIT 1500000; Profile altered.
```

The cost for the composite limit is calculated as follows:

Cost = (10 × LOGICAL_READS_PER_SESSION) + (2 × CONNECT_TIME)

If the user has performed 100,000 block reads and was connected for two hours, the cost thus far would be (10 × 100,000) + (2 × 120) = 1,000,240.

The user will be restricted when this cost exceeds 1,500,000 or when the values for LOGICAL_READS_PER_SESSION or CONNECT_TIME set in the profile are reached.

<u>**Password Verification Function**</u>

You can create a function to verify the complexity of the passwords and assign the function name to the PASSWORD_VERIFY_FUNCTION parameter in the profile. When a password is changed, Oracle checks to see whether the supplied password satisfies the conditions specified in this function. Oracle provides a default verification function, known as VERIFY_FUNCTION, which is in the rdbms/admin directory of your Oracle software installation; the script is named utlpwdmg.sql. The password verification function should be owned by SYS and should have the following characteristics.

```
FUNCTION SYS.<function_name>

( <userid_variable> IN VARCHAR2 (30),

<password_variable> IN VARCHAR2 (30),
<old_password_variable> IN VARCHAR2 (30) )  RETURN BOOLEAN
```

Oracle's default password verification function checks that the password conforms to the following:

- Is not the same as the username
- Has a minimum length
- Is not too simple; a list of words is checked
- Contains at least one letter, one digit, and one punctuation mark
- Differs from the previous password by at least three letters

If the new password satisfies all the conditions, the function returns a Boolean result of TRUE, and the user's password is changed.

Altering Profiles

Using the ALTER PROFILE command changes profile values. You can change any parameter in the profile using this command. The changes take effect the next time the user connects to the database. For example, to add a password verification function and set a composite limit to the profile you created in the previous example, use the following:

```
SQL> ALTER PROFILE ACCOUNTING_USER LIMIT

PASSWORD_VERIFY_FUNCTION VERIFY_FUNCTION

COMPOSITE_LIMIT 1500;



Profile altered.
```

Dropping Profiles

To drop a profile, you use the DROP PROFILE command. If any user is assigned the profile you want to drop, Oracle returns an error. You can drop such profiles by specifying CASCADE, in which case the users who have that profile will be assigned the DEFAULT profile.

```
SQL> DROP PROFILE ACCOUNTING_USER CASCADE;

Profile dropped.
```

Assigning Profiles

To assign profiles to users, you use the CREATE USER or ALTER USER command. These commands are discussed later. This example assigns the ACCOUNTING_USER profile to an existing user named SCOTT:

```
SQL> ALTER USER SCOTT

  PROFILE ACCOUNTING_USER;  User altered.
```

Querying Profile Information

You can query profile information from the DBA_PROFILES view. The following example shows information about the profile created previously. The RESOURCE_TYPE column indicates whether the parameter is KERNEL (resource) or PASSWORD.

```
SQL> SELECT RESOURCE_NAME, LIMIT

FROM DBA_PROFILES

WHERE PROFILE = 'ACCOUNTING_USER'

AND RESOURCE_TYPE = 'KERNEL';


RESOURCE_NAME                    LIMIT

-------------------------------- ----------

COMPOSITE_LIMIT                  1500

SESSIONS_PER_USER                6

CPU_PER_SESSION                  DEFAULT

CPU_PER_CALL                     DEFAULT

LOGICAL_READS_PER_SESSION        DEFAULT
LOGICAL_READS_PER_CALL           10000000

IDLE_TIME                        120

CONNECT_TIME                     UNLIMITED

PRIVATE_SGA                      DEFAULT

9 rows selected.
```

www.csitportal.com

The view USER_RESOURCE_LIMITS shows the limit defined for the current user for resource, and the view USER_PASSWORD_LIMITS shows the limit defined for the password.

Users

Access to the Oracle database is provided using database accounts known as usernames (users). If the user owns database objects, the account is known as a *schema*, which is a logical grouping of all the objects owned by the user. Persons requiring access to the database should have a valid username created in the database. The following properties are associated with a database user account:

Authentication method: Each user must be authenticated to connect to the database by using a password, through the operating system, or via the Enterprise Directory Service. Operating system authentication is discussed in the "Privileges and Roles" section.

Default and temporary tablespaces: The default tablespace specifies a tablespace for the user to create objects if another tablespace is not explicitly specified. The user needs a quota assigned in the tablespace to create objects, even if the tablespace is the user's default. You use the temporary tablespace to create the temporary segments; the user need not have any quota assigned in this tablespace.

Space quota The user needs a *space quota* assigned in each tablespace in which they want to create the objects. By default, a newly created user does not have any space quota allocated on any tablespace to create schema objects. For the user to create schema objects such as tables or materialized views, you must allocate a space quota in tablespaces.

Profile The user can have a profile to specify the resource limits and password settings. If you don't specify a profile when you create the user, the DEFAULT profile is assigned.

NOTE: When you create the database, the SYS and SYSTEM users are created. SYS is the schema that owns the data dictionary.

Managing Users

To create users in the database, you use the CREATE USER command. Specify the authentication method when you create the user. A common authentication method is using the database; the username is assigned a password, which is stored encrypted in the database. Oracle verifies the password when establishing a connection to the database.

As an example, let's create a user JOHN with the various clauses available in the CREATE USER command.

```
SQL> CREATE USER JOHN
```

```
IDENTIFIED BY “B1S2!”
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP
QUOTA UNLIMITED ON USERS
QUOTA 1M ON INDX
PROFILE ACCOUNTING_USER
PASSWORD EXPIRE
ACCOUNT UNLOCK User created.
```

The IDENTIFIED BY clause specifies that the user will be authenticated using the database. To authenticate the user using the operating system, specify IDENTIFIED EXTERNALLY. The password specified is not case sensitive.

The user JOHN can connect to the database only if he has the CREATE SESSION privilege. Granting privileges and roles is discussed later, in the section “Privileges and Roles.” The CREATE SESSION privilege is granted to user JOHN by specifying the following:

```
SQL> GRANT CREATE SESSION TO JOHN;

Grant succeeded.
```

NOTE: To create extents, a user with the UNLIMITED TABLESPACE system privilege does not need any space quota in any tablespace.

Modifying User Accounts

You can modify all the characteristics you specified when creating a user by using the ALTER USER command. You can also assign or modify the default roles assigned to the user (discussed later on). Changing the default tablespace of a user affects only the objects created in the future.

The following example changes the default tablespace of JOHN and assigns a new password.

```
ALTER USER JOHN IDENTIFIED BY SHADOW2#

DEFAULT TABLESPACE APPLICATION_DATA;
```

You can lock or unlock a user’s account as follows:

ALTER USER <username> ACCOUNT [LOCK/UNLOCK]

You can also expire the user’s password:

ALTER USER <username> PASSWORD EXPIRE

Users must change the password the next time they log in, or you must change the password. If the password is expired, SQL*Plus prompts for a new password at login time.

In the following example, setting the quota to 0 revokes the tablespace quota assigned. The objects created by the user in the tablespace remain there, but no new extents can be allocated in that tablespace.

ALTER USER JOHN QUOTA 0 ON USERS;

Dropping Users

You can drop a user from the database by using the DROP USER command. If the user (schema) owns objects, Oracle returns an error. If you specify the CASCADE keyword, Oracle drops all the objects owned by the user and then drops the user. If other schema objects, such as procedures, packages, or views, refer to the objects in the user’s schema, they become invalid. When you drop objects, space is freed up immediately in the relevant tablespaces.

The following example shows how to drop the user JOHN, with all the owned objects.

DROP USER JOHN CASCADE;

WARNING: You cannot drop a user who is currently connected to the database.

Authenticating Users

In this section, we will discuss two widely used user-authenticating methods:

- Authentication by the database
- Authorization by the operating system

When you use database authentication, you define a password for the user (the user can change the password), and Oracle stores the password in the database (encrypted). When users connect to the database, Oracle compares the password supplied by the user with the password in the database.

When you use authorization by the operating system, Oracle verifies the operating system login account and connects to the database—users need not specify a username and password.

Oracle does not store the passwords of such operating-system authenticated users, but they must have a username in the database. The initialization parameter OS_AUTHENT_PREFIX determines the prefix used for operating system authorization. By default, the value is OPS$. For example, if your operating system login name is ALEX, the database username should be

OPS$ALEX. When Alex specifies CONNECT / or does not specify a username to connect to the database, Oracle tries to connect Alex to the OPS$ALEX account. You can set the OS_AUTHENT_PREFIX parameter to a null string “”; this will not add any prefix. To create an operating-system authenticated user, use the following:

```
SQL> CREATE USER OPS$ALEX IDENTIFIED EXTERNALLY;
```

To connect to a remote database using operating system authorization, set the REMOTE_OS_AUTHENT parameter to TRUE. You must be careful in using this parameter, because connections can be made from any computer.

For example, if you have an operating system account named ORACLE and a database account OPS$ORACLE, you can connect to the database from the machine where the database resides. If you set REMOTE_OS_AUTHENT to TRUE, you can log in to any server with the ORACLE operating system account and connect to the database over the network. If a user creates an operating system ID named ORACLE and is on the network, the user can connect to the database using operating system authorization.

Querying User Information

You can query user information from the data dictionary views DBA_USERS and USER_USERS. USER_USERS shows only one row: information about the current user. You can obtain the user account status, password expiration date, account locked date (if locked), encrypted password, default and temporary tablespaces, profile name, and creation date from this view.

Oracle creates a numeric ID and assigns it to the user when the user is created. SYS has an ID of 0.

```
SQL> SELECT USERNAME, DEFAULT_TABLESPACE,

TEMPORARY_TABLESPACE, PROFILE,

ACCOUNT_STATUS, EXPIRY_DATE

FROM DBA_USERS

WHERE USERNAME = ‘JOHN’;
```

The view ALL_USERS shows the username and creation date.

```
SQL> SELECT * FROM ALL_USERS;
```

Managing Privileges

In the Oracle database, *privileges* control access to the data and restrict the actions users can perform. Through proper privileges, users can create, drop, or modify objects in their own schema or in another user's schema. Privileges also determine the data to which a user should have access. You can grant privileges to a user by means of two methods:

- You can assign privileges directly to the user.
- You can assign privileges to a role, and then assign the role to the user.

A *role* is a named set of privileges, which eases the management of privileges. For example, if you have 10 users needing access to the data in the accounting tables, you can grant the privileges required to a role and grant the role to the 10 users. There are two types of privileges:

Object privileges *Object privileges* are granted on schema objects that belong to a different schema. The privilege can be on data (to read, modify, delete, add, or reference), on a program (to execute), or on an object (to change the structure).

System privileges *System privileges* provide the right to perform a specific action on any schema in the database. System privileges do not specify an object, but are granted at the database level. Certain system privileges are very powerful and should be granted only to trusted users. System privileges and object privileges can be granted to a role.

PUBLIC is a user group defined in the database; it is not a database user or a role. Every user in the database belongs to this group. Therefore, if you grant privileges to PUBLIC, they are available to all users of the database.

NOTE: A user and a role cannot have the same name.

Object Privileges

Object privileges are granted on a specific object. The owner of the object has all privileges on the object. The owner can grant privileges on that object to any other users of the database. The owner can also authorize another user in the database to grant privileges on the object to other users. For example, user JOHN owns a table named CUSTOMER and grants read and update privileges to JAMES. (To specify multiple privileges, separate them with a comma.)

```
SQL> GRANT SELECT, UPDATE ON CUSTOMER TO JAMES;
```

JAMES cannot insert into or delete from CUSTOMER; JAMES can only query and update rows in the CUSTOMER table. JAMES cannot grant the privilege to another user in the database, because JAMES is not authorized by JOHN to do so. If the privilege is granted with the WITH GRANT OPTION, JAMES can grant the privilege to others.

```
SQL> GRANT SELECT, UPDATE ON CUSTOMER

TO JAMES WITH GRANT OPTION;
```

The INSERT, UPDATE, or REFERENCES privileges can be granted on columns also. For example:

```
SQL> GRANT INSERT (CUSTOMER_ID) ON CUSTOMER TO JAMES;
```

The following are the object privileges that can be granted to users of the database:

SELECT Grants read (query) access to the data in a table, view, sequence, or materialized view.

UPDATE Grants update (modify) access to the data in a table, column, view, or materialized view.

DELETE Grants delete (remove) access to the data in a table, view, or materialized view.

INSERT Grants insert (add) access to a table, column, view, or materialized view.

EXECUTE Grants execute (run) privilege on a PL/SQL stored object, such as a procedure, package, or function.

READ Grants read access on a directory.

INDEX Grants index creation privilege on a table.

REFERENCES Grants reference access to a table or columns to create foreign keys that can reference the table.

ALTER Grants access to modify the structure of a table or sequence.

ON COMMIT REFRESH Grants the privilege to create a refresh-on-commit materialized view on the specified table.

QUERY REWRITE Grants the privilege to create a materialized view for query rewrite using the specified table.

WRITE Allows the external table agent to write a log file or a bad file to the directory. This privilege is associated only with the external tables.

UNDER Grants the privilege to create a sub-view under a view.

The following are some points related to object privileges that you need to remember:

- Object privileges can be granted to a user, a role, or PUBLIC.
- If a view refers to tables or views from another user, you must have the privilege WITH GRANT OPTION on the underlying tables of the view to grant any privilege on the view to another user. For example, JOHN owns a view, which references a table from JAMES. To grant the SELECT privilege on the view to another user, JOHN should have received the SELECT privilege on the table WITH GRANT OPTION.
- Any object privilege received on a table provides the grantee the privilege to lock the table.
- The SELECT privilege cannot be specified on columns; to grant column-level SELECT privileges, create a view with the required columns and grant SELECT on the view.
- You can specify ALL or ALL PRIVILEGES to grant all available privileges on an

object (for example, GRANT ALL ON CUSTOMER TO JAMES). Even if you have the DBA privilege, to grant privileges on objects owned by another user you must have been granted the appropriate privilege on the object WITH GRANT OPTION.

- Multiple privileges can be granted to multiple users and/or roles in one statement. For example, GRANT INSERT, UPDATE, SELECT ON CUSTOMER TO ADMIN_ROLE, JULIE, SCOTT;

System Privileges

System privileges are the privileges that enable the user to perform an action; they are not specified on any particular object. Like object privileges, system privileges also can be granted to a user, a role, or PUBLIC. There are many system privileges in Oracle; Table 1 summarizes the privileges used to manage objects in the database. The CREATE, ALTER, and DROP privileges provide the ability to create, modify, and drop the object specified in the user's schema.

When a privilege is specified with ANY, the user is authorized to perform the action on any schema in the database. Table 2 shows the types of privileges that are associated with certain types of objects. For example, the SELECT ANY TABLE privilege gives the user the ability to query all tables or views in the database, regardless of who owns them; the SELECT ANY SEQUENCE privilege gives the user the ability to select from all sequences in the database.

Here are some points to remember about system privileges:

- To connect to the database, you need the CREATE SESSION privilege.
- To truncate a table that belongs to another schema, you need the DROP ANY TABLE privilege.
- The CREATE ANY PROCEDURE (or EXECUTE ANY PROCEDURE) privilege allows the user to create, replace, or drop (or execute) procedures, packages, and functions; this includes Java classes.
- The CREATE TABLE privilege gives you the ability to create, alter, drop, and query tables in a schema.
- SELECT, INSERT, UPDATE, and DELETE are object privileges, but SELECT ANY, INSERT ANY, UPDATE ANY, and DELETE ANY are system privileges (in other words, they do not apply to a particular object).

Granting System Privileges

System privileges are also granted to a user, a role, or PUBLIC by using the GRANT command. The WITH ADMIN OPTION clause gives the grantee the privilege to grant the privilege to another user, role, or PUBLIC. For example, if JOHN needs to create a table under JAMES's schema, he needs the CREATE ANY TABLE privilege. This privilege not only allows JOHN to create a table under JAMES's schema, but also allows the creation of a table under any schema in the database.

```
SQL> GRANT CREATE ANY TABLE TO JOHN;
```

If John must be able to grant this privilege to others, he should be granted the privilege with the WITH ADMIN OPTION clause (or should have the GRANT ANY PRIVILEGE privilege).

```
SQL> GRANT CREATE ANY TABLE TO JOHN WITH ADMIN OPTION;
```

Revoking Privileges

You can revoke a user's object privileges and system privileges by using the REVOKE statement. You can revoke a privilege if you have granted it to the user or if you have been granted that privilege with the WITH ADMIN OPTION (for system privileges) or the WITH GRANT OPTION (for object privileges) clauses. Here are some examples of revoking privileges.

To revoke the UPDATE privilege granted to JAMES from JOHN on JOHN's CUSTOMER table, use the following:

```
SQL> REVOKE UPDATE ON CUSTOMER FROM JAMES;
```

To revoke the SELECT ANY TABLE and CREATE TRIGGER privileges granted to JULIE, use the following:

SQL> REVOKE SELECT ANY TABLE, CREATE TRIGGER

FROM JULIE;

The following statement revokes all the privileges granted by JAMES on the STATE table to JULIE. JAMES executes this statement.

SQL> REVOKE ALL ON STATE FROM JULIE;

Keep the following in mind when revoking privileges:

- If multiple users (or administrators) have granted an object privilege to a user, revoking the privilege by one administrator will not prevent the user from performing the action, because the privileges granted by the other administrators are still valid.
- To revoke the WITH ADMIN OPTION or WITH GRANT OPTION, you must revoke the privilege and re-grant the privilege without the clause.
- You cannot selectively revoke column privileges; you must revoke the privileges from the table and grant them again with the appropriate columns.
- If a user has used their system privileges to create or modify an object, and subsequently the user's privilege is revoked, no change is made to the objects that the user has already created or modified. The user just can no longer create or modify the object.
- If a PL/SQL program or view is created based on an object privilege (or a DML system privilege such as SELECT ANY, UPDATE ANY, and so on), revoking the privilege will invalidate the object.
- If user A is granted a system privilege WITH ADMIN OPTION, and grants the privilege to user B, user B's privilege still remains when user A's privilege is revoked.
- If user A is granted an object privilege WITH GRANT OPTION, and grants the privilege to user B, user B's privilege is also automatically revoked when user A's privilege is revoked, and the objects that use the privileges under user A and user B are invalidated.

Querying Privilege Information

You can query privilege information from the data dictionary by using various views. Table 3 lists and describes the views that provide information related to privileges.

View Name	Description
ALL_TAB_PRIVS DBA_TAB_PRIVS USER_TAB_PRIVS	Lists the object privileges. ALL_TAB_PRIVS shows only the privileges granted to the user and to PUBLIC.

www.csitportal.com

ALL_TAB_PRIVS_MADE USER_TAB_PRIVS_MADE	Lists the object grants made by the current user or grants made on the objects owned by the current user.
ALL_TAB_PRIVS_RECD USER_TAB_PRIVS_RECD	Lists the object grants received by the current user or PUBLIC.
ALL_COL_PRIVS DBA_COL_PRIVS USER_COL_PRIVS	Lists column privileges.
ALL_COL_PRIVS_MADE USER_COL_PRIVS_MADE	Lists column privileges made by the current user.
ALL_COL_PRIVS_RECD USER_COL_PRIVS_RECD	Lists column privileges received by the current user.
DBA_SYS_PRIVS USER_SYS_PRIVS	Lists system privilege information.
SESSION_PRIVS	Lists the system privileges available for the current session.

Managing Roles

A *role* is a named group of privileges that you can use to ease the administration of privileges. For example, if your accounting department has 30 users and all need similar access to the tables in the accounts receivable application, you can create a role and grant the appropriate system and object privileges to the role. You can grant the role to each user of the accounting department, instead of granting each object and system privilege to individual users.

Using the CREATE ROLE command creates the role. No user owns the role; it is owned by the database. When a role is created, no privileges are associated with it. You must grant the appropriate privileges to the role. For example, to create a role named ACCTS_RECV and grant certain privileges to the role, use the following:

```
CREATE ROLE ACCTS_RECV;

GRANT SELECT ON GENERAL_LEDGER TO ACCTS_RECV;

GRANT INSERT, UPDATE ON JOURNAL_ENTRY TO ACCTS_RECV;
```

Similar to users, roles can also be authenticated. The default is NOT IDENTIFIED, which means no authorization is required to enable or disable the role. The following authorization methods are available:

Database Using a password associated with the role, the database authorizes the role. Whenever such roles are enabled, the user is prompted for a password if the role is not one of the default roles for the user. In the following example, a role ACCOUNTS_MANAGER is created with a password.

```
SQL> CREATE ROLE ACCOUNTS_MANAGER IDENTIFIED BY ACCMGR;
```

Operating system The role is authorized by the operating system. This is useful when the operating system can associate its privileges with the application privileges, and information about each user is configured in operating system files. To enable operating system role authorization, set the parameter OS_ROLES to TRUE. The following example creates a role, authorized by the operating system.

```
SQL> CREATE ROLE APPLICATION_USER IDENTIFIED EXTERNALLY;
```

You can change the role's password or authentication method by using the ALTER ROLE command. You cannot rename a role. For example:

```
SQL> ALTER ROLE ACCOUNTS_MANAGER IDENTIFIED BY MANAGER;
```

To drop a role, use the DROP ROLE command. Oracle will let you drop a role even if it is granted to users or other roles. When you drop a role, it is immediately removed from the users' role lists.

```
SQL> DROP ROLE ACCOUNTS_MANAGER;
```

Using Predefined Roles

When you create the database, Oracle creates six predefined roles. These roles are defined in the sql.bsq script, which is executed when you run the CREATE DATABASE command. The following roles are predefined:

CONNECT Privilege to connect to the database, to create a cluster, a database link, a sequence, a synonym, a table, and a view, and to alter a session.

RESOURCE Privilege to create a cluster, a table, and a sequence, and to create programmatic objects such as procedures, functions, packages, indextypes, types, triggers, and operators.

DBA All system privileges with the ADMIN option, so the system privileges can be granted to other users of the database or to roles.

SELECT_CATALOG_ROLE Ability to query the dictionary views and tables.

EXECUTE_CATALOG_ROLE Privilege to execute the dictionary packages (SYS- owned packages).

DELETE_CATALOG_ROLE Ability to drop or re-create the dictionary packages.

Also, when you run the catproc.sql script as part of the database creation, the script executes catexp.sql, which creates two more roles:

EXP_FULL_DATABASE Ability to make full and incremental exports of the database using the Export utility.

IMP_FULL_DATABASE Ability to perform full database imports using the Import utility. This is a very powerful role.

Removing Roles

You can remove roles from the database using the DROP ROLE statement. When you drop a role, all privileges that users had through the role are lost. If they used the role to create objects in the database or to manipulate data, those objects and changes remain in the database. To drop a role named HR_UPDATE, use the following statement:

```
DROP ROLE HR_UPDATE;
```

To drop a role, you must have been granted the role with the ADMIN OPTION, or you must have the DROP ANY ROLE system privilege.

Enabling and Disabling Roles

If a role is not the default role for a user, it is not enabled when the user connects to the database. You use the ALTER USER command to set the default roles for a user. You can use the DEFAULT ROLE clause with the ALTER USER command in four ways, as illustrated in the following examples.

To specify the named roles CONNECT and ACCOUNTS_MANAGER as default roles, use the following:

```
ALTER USER JOHN DEFAULT ROLE CONNECT, ACCOUNTS_MANAGER;
```

To specify all roles granted to the user as the default, use the following:

```
ALTER USER JOHN DEFAULT ROLE ALL;
```

To specify all roles except certain roles as the default, use the following:

```
ALTER USER JOHN DEFAULT ROLE ALL EXCEPT RESOURCE,

ACCOUNTS_ADMIN;
```

To specify no roles as the default, use the following:

```
ALTER USER JOHN DEFAULT ROLE NONE;
```

You can specify only roles granted to the user as default roles. The DEFAULT ROLE clause is not available in the CREATE USER command. Default roles are enabled when the user connects to the database and do not require a password.

You enable or disable roles using the SET ROLE command. You specify the maximum number of roles that can be enabled in the initialization parameter MAX_ENABLED_ROLES (the default is 20). You can enable or disable only roles granted to the user. If a role is defined with a password, you must supply the password when you enable the role. For example:

```
SET ROLE ACCOUNTS_ADMIN IDENTIFIED BY MANAGER;
```

To enable all roles, specify the following:

```
SET ROLE ALL;
```

To enable all roles, except the roles specified, use the following:

```
SET ROLE ALL EXCEPT RESOURCE, ACCOUNTS_USER;
```

To disable all roles, including the default roles, use the following:

```
SET ROLE NONE;
```

Querying Role Information

The data dictionary view DBA_ROLES lists the roles defined in the database. The column PASSWORD specifies the authorization method.

```
SQL> SELECT * FROM DBA_ROLES;

SQL> SELECT * FROM DBA_ROLE_PRIVS

WHERE GRANTEE = ‘JOHN’;
```

The view ROLE_ROLE_PRIVS lists the roles granted to the roles, ROLE_SYS_PRIVS lists the system privileges granted to roles, and ROLE_TAB_PRIVS shows information on the object privileges granted to roles.

```
SQL> SELECT * FROM ROLE_ROLE_PRIVS

WHERE ROLE = ‘DBA’;


SQL> SELECT * FROM ROLE_SYS_PRIVS

WHERE ROLE = ‘CONNECT’;

SQL> SELECT * FROM ROLE_TAB_PRIVS

WHERE TABLE_NAME = ‘CUSTOMER’;
```

What is PL/SQL?

PL/SQL stands for **Procedural Language** extension of SQL.

PL/SQL is a combination of SQL along with the procedural features of programming languages.

It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

The PL/SQL Engine:

Oracle uses a **PL/SQL** engine to processes the PL/SQL statements. A PL/SQL language code can be stored in the client system (client-side) or in the database (server-side).

A Simple PL/SQL Block:

Each PL/SQL program consists of SQL and PL/SQL statements which from a PL/SQL block.

PL/SQL Block consists of three sections:

- The Declaration section (optional).
- The Execution section (mandatory).
- The Exception Handling (or Error) section (optional).

Declaration Section:

The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE. This section is optional and is used to declare any placeholders like variables, constants, records and cursors, which are used to manipulate data in the execution section. Placeholders may be any of Variables, Constants and Records, which stores data temporarily. Cursors are also declared in this section.

Execution Section:

The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END. This is a mandatory section and is the section where the program logic is written to perform any task. The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

Exception Section:

The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION. This section is optional. Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully. If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors.

Every statement in the above three sections must end with a semicolon; PL/SQL blocks can be nested within other PL/SQL blocks. Comments can be used to document code.

How a Sample PL/SQL Block Looks

```
DECLARE
     Variable declaration
BEGIN
     Program Execution
EXCEPTION
     Exception handling
END;
```

Advantages of PL/SQL

- ***Block Structures*:** PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.
- ***Procedural Language Capability*:** PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).
- ***Better Performance*:** PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
- ***Error Handling*:** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

PL/SQL Placeholders

Placeholders are temporary storage area. PL/SQL Placeholders can be any of Variables, Constants and Records. Oracle defines placeholders to store data temporarily, which are used to manipulate data during the execution of a PL SQL block.

Define PL/SQL Placeholders

Depending on the kind of data you want to store, you can define placeholders with a name and a datatype. Few of the datatypes used to define placeholders are as given below. Number (n,m) , Char (n) , Varchar2 (n) , Date , Long , Long raw, Raw, Blob, Clob, Nclob, Bfile

PL/SQL Variables

These are placeholders that store the values that can change through the PL/SQL Block.

General Syntax to declare a variable is

variable_name datatype [NOT NULL := value];

- *variable_name* is the name of the variable.
- *datatype* is a valid PL/SQL datatype.
- NOT NULL is an optional specification on the variable.

- *value* or DEFAULT *value*is also an optional specification, where you can initialize a variable.
- Each variable declaration is a separate statement and must be terminated by a semicolon.

For example, if you want to store the current salary of an employee, you can use a variable.

```
DECLARE
salary  number (6);
```

* “salary” is a variable of datatype number and of length 6.

When a variable is specified as NOT NULL, you must initialize the variable when it is declared.

For example: The below example declares two variables, one of which is a not null.

```
DECLARE
salary number(4);
dept varchar2(10) NOT NULL := “HR Dept”;
```

The value of a variable can change in the execution or exception section of the PL/SQL Block. We can assign values to variables in the two ways given below.

1)We can directly assign values to variables.
The General Syntax is:

```
 variable_name:=  value;
```

2) We can assign values to variables directly from the database columns by using a SELECT.. INTO statement. The General Syntax is:

```
SELECT column_name
INTO variable_name
FROM table_name
[WHERE condition];
```

Example: The below program will get the salary of an employee with id '1116' and display it on the screen.

```
DECLARE
 var_salary number(6);
 var_emp_id number(6) = 1116;
BEGIN
 SELECT salary
 INTO var_salary
 FROM employee
 WHERE emp_id = var_emp_id;
 dbms_output.put_line(var_salary);
 dbms_output.put_line('The employee '
            || var_emp_id || ' has  salary  ' || var_salary);
END;
/
```

NOTE: The backward slash '/' in the above program indicates to execute the above PL/SQL Block.

Scope of PS/SQL Variables

PL/SQL allows the nesting of Blocks within Blocks i.e, the Execution section of an outer block can contain inner blocks. Therefore, a variable which is accessible to an outer Block is also accessible to all nested inner Blocks. The variables declared in the inner blocks are not accessible to outer blocks. Based on their declaration we can classify variables into two types.

- *Local* variables - These are declared in a inner block and cannot be referenced by outside Blocks.
- *Global* variables - These are declared in a outer block and can be referenced by its itself and by its inner blocks.

For Example: In the below example we are creating two variables in the outer block and assigning thier product to the third variable created in the inner block. The variable 'var_mult' is declared in the inner block, so cannot be accessed in the outer block i.e. it cannot be accessed after line 11. The variables 'var_num1' and 'var_num2' can be accessed anywhere in the block.

```
1> DECLARE
2>  var_num1 number;
3>  var_num2 number;
4> BEGIN
5>  var_num1 := 100;
6>  var_num2 := 200;
7>  DECLARE
8>   var_mult number;
9>   BEGIN
10>    var_mult := var_num1 * var_num2;
11>   END;
12> END;
13> /
```

PL/SQL Constants

As the name implies a *constant* is a value used in a PL/SQL Block that remains unchanged throughout the program. A constant is a user-defined literal value. You can declare a constant and use it instead of actual value.

For example: If you want to write a program which will increase the salary of the employees by 25%, you can declare a constant and use it throughout the program. Next time when you want to increase the salary again you can change the value of the constant which will be easier than changing the actual value throughout the program.

General Syntax to declare a constant is:

constant_name CONSTANT datatype := VALUE;

- *constant_name* is the name of the constant i.e. similar to a variable name.
- The word *CONSTANT* is a reserved word and ensures that the value does not change.
- *VALUE* - It is a value which must be assigned to a constant when it is declared. You cannot assign a value later.

For example, to declare salary_increase, you can write code as follows:

```
DECLARE
salary_increase CONSTANT number (3) := 10;
```

You *must* assign a value to a constant at the time you declare it. If you do not assign a value to a constant while declaring it and try to assign a value in the execution section, you will get a error. If you execute the below Pl/SQL block you will get error.

```
DECLARE
 salary_increase CONSTANT number(3);
BEGIN
 salary_increase := 100;
 dbms_output.put_line (salary_increase);
END;
```

Stored Procedures

What is a Stored Procedure?

A **stored procedure** or in simple a **proc** is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.

A procedure has a header and a body. The header consists of the name of the procedure and the parameters or variables passed to the procedure. The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.

A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

Procedures: Passing Parameters

We can pass parameters to procedures in three ways.
1) IN-parameters
2) OUT-parameters
3) IN OUT-parameters

A procedure may or may not return any value.

General Syntax to create a procedure is:

```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters]
IS
   Declaration section
BEGIN
   Execution section
EXCEPTION
  Exception section
END;
```

IS - marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.

The syntax within the brackets [ ] indicate they are optional. By using CREATE OR REPLACE together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.

Procedures: Example

The below example creates a procedure ‘employer_details’ which gives the details of the employee.

www.csitportal.com

```
CREATE OR REPLACE PROCEDURE employer_details
IS
 CURSOR emp_cur IS
 SELECT first_name, last_name, salary FROM emp_tbl;
 emp_rec emp_cur%rowtype;
BEGIN
 FOR emp_rec in sales_cur
 LOOP
 dbms_output.put_line(emp_cur.first_name || ' ' ||emp_cur.last_name
    || ' ' ||emp_cur.salary);
END LOOP;
END;
/
```

How to execute a Stored Procedure?

There are two ways to execute a procedure.

1) From the SQL prompt.

```
 EXECUTE [or EXEC] procedure_name;
```

2) Within another procedure – simply use the procedure name.

```
  procedure_name;
```

Trigger

A trigger is a pl/sql block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement is executed.

Syntax of Triggers

Syntax for Creating a Trigger

```
 CREATE [OR REPLACE ] TRIGGER trigger_name
 {BEFORE | AFTER | INSTEAD OF }
 {INSERT [OR] | UPDATE [OR] | DELETE}
 [OF col_name]
 ON table_name
 [REFERENCING OLD AS o NEW AS n]
 [FOR EACH ROW]
 WHEN (condition)
 BEGIN
   --- sql statements
 END;
```

Types of PL/SQL Triggers

There are two types of triggers based on the which level it is triggered.
1) Row level trigger - An event is triggered for each row upated, inserted or deleted.
2) Statement level trigger - An event is triggered for each sql statement executed.

PL/SQL Trigger Execution Hierarchy

The following hierarchy is followed when a trigger is fired.
1) BEFORE statement trigger fires first.
2) Next BEFORE row level trigger fires, once for each row affected.
3) Then AFTER row level trigger fires once for each affected row. This events will alternates between BEFORE and AFTER row level triggers.
4) Finally the AFTER statement level trigger fires.

What are Cursors?

A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.

This temporary work area is used to store the data retrieved from the database, and manipulate this data. A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the *active* set.

There are two types of cursors in PL/SQL:

Implicit cursors

These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.

Explicit cursors

They must be created when you are executing a SELECT statement that returns more than one row. Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. When you fetch a row the current row position moves to next row.

Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.

```
DECLARE
CURSOR emp_cur IS
SELECT first_name, last_name, salary FROM emp_tbl;
emp_rec emp_cur%rowtype;
BEGIN
```

```
FOR emp_rec in sales_cur
LOOP
dbms_output.put_line(emp_cur.first_name || ' ' ||emp_cur.last_name || ' ' ||emp_cur.salary);
END LOOP;
END;
/
```

PL/SQL Functions

What is a Function in PL/SQL?

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

General Syntax to create a function is

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
RETURN return_datatype;
IS
Declaration_section
BEGIN
Execution_section
Return return_variable;
EXCEPTION
exception section
Return return_variable;
END;
```

1) **Return Type:** The header section defines the return type of the function. The return datatype can be any of the oracle datatype like varchar, number etc.
2) The execution and exception section both should return a value which is of the datatype defined in the header section.

For example, let’s create a frunction called "employer_details_func' similar to the one created in stored proc

```
CREATE OR REPLACE FUNCTION employer_details_func
   RETURN VARCHAR(20);
IS
   emp_name VARCHAR(20);
BEGIN
        SELECT first_name INTO emp_name
        FROM emp_tbl WHERE empID = '100';
        RETURN emp_name;
END;
/
```

In the example we are retrieving the ‘first_name’ of employee with empID 100 to variable ‘emp_name’.
The return type of the function is VARCHAR which is declared in line no 2.
The function returns the 'emp_name' which is of type VARCHAR as the return value in line no 9.

How to execute a PL/SQL Function?

A function can be executed in the following ways.

1) Since a function returns a value we can assign it to a variable.

employee_name := employer_details_func;

If ‘employee_name’ is of datatype varchar we can store the name of the employee by assigning the return type of the function to it.

2) As a part of a SELECT statement

SELECT employer_details_func FROM dual;

3) In a PL/SQL Statements like,

dbms_output.put_line(employer_details_func);
This line displays the value returned by the function.

Package: Package is oracle database object that binds many oracle objects like procedures, cursors, functions etc. It is comprised of two parts:

- Package Specification
- Package Body

THE PACKAGE SPECIFICATION:

```
CREATE OR REPLACE PACKAGE c_package AS
   -- Adds a customer
   PROCEDURE addCustomer(c_id   customers.id%type,
   c_name  customers.name%type,
   c_age  customers.age%type,
   c_addr customers.address%type,
   c_sal  customers.salary%type);
   PROCEDURE delCustomer(c_id  customers.id%TYPE);
   PROCEDURE listCustomer;

END c_package;
/
```

```
Package created.
```

CREATING THE PACKAGE BODY:

```
CREATE OR REPLACE PACKAGE BODY c_package AS
   PROCEDURE addCustomer(c_id  customers.id%type,
      c_name customers.name%type,
      c_age  customers.age%type,
      c_addr  customers.address%type,
      c_sal   customers.salary%type)
   IS
   BEGIN
      INSERT INTO customers (id,name,age,address,salary)
         VALUES(c_id, c_name, c_age, c_addr, c_sal);
   END addCustomer;

   PROCEDURE delCustomer(c_id   customers.id%type) IS
   BEGIN
       DELETE FROM customers
         WHERE id = c_id;
   END delCustomer;
   PROCEDURE listCustomer IS
   CURSOR c_customers is
      SELECT  name FROM customers;
   TYPE c_list is TABLE OF customers.name%type;
   name_list c_list := c_list();
   counter integer :=0;
   BEGIN
      FOR n IN c_customers LOOP
      counter := counter +1;
      name_list.extend;
      name_list(counter)  := n.name;
      dbms_output.put_line('Customer(' ||counter|| ')'||name_list(counter));
      END LOOP;
   END listCustomer;
END c_package;
/
Package body created.
```

USING THE PACKAGE:

The following program uses the methods declared and defined in the package *c_package*.

```
DECLARE
   code customers.id%type:= 8;
BEGIN
      c_package.addcustomer(7, 'Rajnish', 25, 'Chennai', 3500);
      c_package.addcustomer(8, 'Subham', 32, 'Delhi', 7500);
      c_package.listcustomer;
      c_package.delcustomer(code);
      c_package.listcustomer;
END;
/
```

View

An Oracle **VIEW**, in essence, is a virtual table that does not physically exist. Rather, it is created by a query joining one or more tables.

Create VIEW

Syntax

The syntax for the Oracle **CREATE VIEW Statement** is:

```
CREATE VIEW view_name AS
  SELECT columns
  FROM tables
  WHERE conditions;
```

view_name is the name of the Oracle VIEW that you wish to create.

Example

Here is an example of how to use the Oracle CREATE VIEW:

```
CREATE VIEW sup_orders AS
  SELECT suppliers.supplier_id, orders.quantity, orders.price
  FROM suppliers
  INNER JOIN orders
  ON suppliers.supplier_id = orders.supplier_id
  WHERE suppliers.supplier_name = 'Microsoft';
```

This Oracle CREATE VIEW example would create a virtual table based on the result set of the SELECT statement. You can now query the Oracle VIEW as follows:

```
SELECT *
FROM sup_orders;
```

Update VIEW

You can modify the definition of an Oracle VIEW without dropping it by using the Oracle CREATE OR REPLACE VIEW Statement.

Syntax

The syntax for the Oracle **CREATE OR REPLACE VIEW Statement** is:

```
CREATE OR REPLACE VIEW view_name AS
  SELECT columns
  FROM table
  WHERE conditions;
```

Example

Here is an example of how you would use the Oracle CREATE OR REPLACE VIEW Statement:

```
CREATE or REPLACE VIEW sup_orders AS
  SELECT suppliers.supplier_id, orders.quantity, orders.price
  FROM suppliers
  INNER JOIN orders
  ON suppliers.supplier_id = orders.supplier_id
  WHERE suppliers.supplier_name = 'Apple';
```

This Oracle CREATE OR REPLACE VIEW example would update the definition of the Oracle VIEW called *sup_orders* without dropping it. If the Oracle VIEW did not yet exist, the VIEW would merely be created for the first time.

Drop VIEW

Once an Oracle VIEW has been created, you can drop it with the Oracle DROP VIEW Statement.

Syntax

The syntax for the Oracle **DROP VIEW Statement** is:

```
DROP VIEW view_name;
```

view_name is the name of the view that you wish to drop.

Example

Here is an example of how to use the Oracle DROP VIEW Statement:

```
DROP VIEW sup_orders;
```

BACKUP and RECOVERY

BACKUP AND RECOVERY CONCEPTS

Backup and recovery is based on a threefold recovery methodology consisting of: (1) exports and imports, (2) normal backups, and (3) the use of archive logging of redo logs.

Types of Database Failure

The common types of failure are:

Statement failure: This failure is caused by an error in an Oracle program.

Process failure: This is a user process failure such as abnormal disconnection or process termination (someone clicks the close button on a telnet session when they did not intend to disconnect).

Instance failure: Some problem prevents the Oracle instance from functioning - lack of SGA memory to be allocated or failure of a background process.

User or application error: User accidentally deletes data that was not to be deleted; or the application program causes a similar error.

Media failure: A physical problem such as a disk head crash that causes the loss of data on a disk drive.

What is Backup and Recovery?
Backup means to create a means for recovering a database from disaster.
There are two types of backups: physical and logical.
Physical backup is the creation of copies of critical physical database files.
Logical backup is the use of the Oracle Data Pump Export or Oracle Export utility to extract specific data and to store that data to an export binary file.
A physical backup can be made by using either the Recovery Manager utility program or operating system utilities such as the UNIX cp (copy) command to copy files to a backup location.
Restore -- reconstruct an Oracle database by copying backup files to the original file locations of an Oracle database.
Recover -- update a restored datafile by applying redo records from redo logs to bring a database back to the point in time where a failure occurred.

When a database is recovered, it is first restored from a physical backup, then redo logs are used to roll forward to the point of failure. This is illustrated in the figure shown here.
The use of Oracle's Recovery Manager (RMAN) utility also enables you to recover restored datafiles using incremental backups, which are backups of a datafile that contain only blocks that changed after the last backup.

Crash recovery/instance recovery are performed by Oracle automatically after an instance fails and is restarted.

Instance Recovery

Instance recovery is an automatic procedure that includes two operations:
· Rolling forward the backup to a more current time by applying online redo records.

· Rolling back all changes made in uncommitted transactions to their original state.

Media Recovery

Media recovery (replacement of a failed hard drive, for example) requires the DBA to use recovery commands.

- The SQLPLUS commands RECOVER or ALTER DATABASE RECOVER are used to apply archived redo logs to datafiles being recovered.
- Use this approach to recover a lost data file:
- Copy the lost file from the previous physical backup using operating system commands such as the UNIX cp command.
- Open the database to the mount stage and issue the ALTER DATABASE RECOVER command.
- Following this, alter the database to the open stage: ALTER DATABASE OPEN.

System Change Number

The SCN (system change number) is an ever-increasing internal timestamp. Oracle uses this to identify a committed version of the database.

- Each new committed transaction requires Oracle to record a new SCN.
- The SCN can be used to perform an incomplete recovery to a specific point in time.
- The SCN is displayed in the alert log file.
- Each control file, datafile header, and redo log record stores an SCN.
- The redo log files have a log sequence number, a low SCN, and a high SCN.
- The low SCN records the lowest SCN in the log file and the high SCN records the highest SCN in the log file.

Archive Logs

Redo logs store all transactions that alter the database, all committed updates, adds, deletes of tables, structures, or data.

- When data changes are made to Oracle tables, index, and other objects, Oracle records both the original and new values of the objects to the redo log buffer in memory. This is a redo record.
- Oracle records both committed and uncommitted changes in redo log buffers.
- The redo log buffer records are written to the online redo log file (see earlier notes for details on this activity).
- Recall there are at least two online redo log file groups used in a circular fashion.

When archiving is disabled, only data in the current offline and online redo logs can be recovered. When the system recycles through all redo logs, old ones are reused destroying the contents of earlier database modifications.

When archiving is enabled, redo logs are written out to storage before reuse allowing recovery to a specific point in time since the last full cold backup.

Under Oracle redo logs are specified in groups, each group is archived together.

Redo logs cannot be used to recover a database brought back from a full export.

Simple Backup and Recovery Strategy

There are just a few basic principles you need to follow for an effective backup and recovery strategy. These are:
Maintain multiple copies of the online redo logs (run multiplexed copies on different disks).

1. Archive the redo logs to multiple locations or make frequent backups of your archived redo logs.
2. Maintain multiple, concurrent copies of your control file using Oracle multiplexing in conjunction with operating system mirroring.
3. Backup datafiles (these files include all tablespaces), control files, and archived redo logs frequently (but not the online redo log files). Optionally, backup the init.ora and config.ora files. Store them in a safe place.

Undo Segments

Undo segments store information about a data block before it is changed.

- These old data values represent data that have been uncommitted (not written to a datafile by DBWn).
- Oracle uses undo segment information during database recovery to undo uncommitted changes that are applied from the redo log files to the datafiles.
- Thus the redo log file records are applied to the datafiles during recovery, then the undo segments are used to undo uncommitted changes.

Using Online Redo Log Files

A typical type of failure is a power outage.

- In this case, Oracle is prevented from writing data from the database buffer cache to the datafiles.
- Recall, however, that LGWR did write redo log records of committed changes to the redo log files.

- The old version of datafiles can be combined with changes in the online and archived redo log files to reconstruct data that was lost during the power outage.

Logical Backups (Exports)

A logical backup involves reading a set of database records and writing them to a file.

- The Data Pump Export utility is used for this type of backup. This is commonly termed an export.
- The Data Pump Import utility is used to recover data generated by the export. This is commonly termed an import.

The Data Pump Export and Data Pump Import utilities are meant to replace the Export and Import utilities provided with earlier versions of Oracle. However, the Export and Import utilities are still available. We will discuss both of these.

DATA PUMP EXPORT AND IMPORT UTILITIES

Data Pump Export Utility

- This utility queries the database including the data dictionary and writes output to an XML file called an export dump file.
- Export capabilities include:
    - ✓ Full database.
    - ✓ Specific users.
    - ✓ Specific tablespaces.
    - ✓ Specific tables.
    - ✓ Ability to specify whether to export grants, indexes, and constraints associated with tables.
- Export dump file contains commands needed to recreate all selected objects and data completely.
- Data Pump export dump files are NOT compatible with files created by earlier versions of the Export utility (9i and earlier).

Data Pump Import Utility

- Reads an export dump file and executes any commands found there.
- Import capabilities include:
    - ✓ Can import data into same database or a different database.
    - ✓ Can import data into the same or a different schema.

- ✓ Can import selected data.

Using the Data Pump Export and Import

Data Pump runs as a server process. This provides the following performance advantages:

- ➢ Client processes used to start a job can disconnect and later reattach to the job.
- ➢ Performance is enhanced because data no longer has to be processed by a client program (the old export/import utility programs).
- ➢ Data Pump extractions can be parallelized.

Data Pump requires the DBA to create directories for the datafiles and log files it creates.

- ➢ Requires the CREATE ANY DIRECTORY privilege, and the external directory must already exist.
- ➢ Use the CREATE DIRECTORY command to create a directory pointer within Oracle to the external directory to be used.
- ➢ Write/read privileges are required for this directory.

Data Pump Export Options

The utility named expdp serves as the interface to Data Pump.
This utility has various command-line input parameters to specify characteristics of an export job when one is created. This table shows the parameters for the expdp utility.

Keyword	Description (Default)
ATTACH	Attach to existing job, e.g. ATTACH [=job name].
CONTENT	Specifies data to unload where the valid keywords are: (ALL), DATA_ONLY, and METADATA_ONLY.
DIRECTORY	Directory object to be used for dumpfiles and logfiles.
DUMPFILE	List of destination dump files (expdat.dmp), e.g. DUMPFILE=scott1.dmp, scott2.dmp, dmpdir:scott3.dmp.
ESTIMATE	Calculate job estimates where the valid keywords are: (BLOCKS) and STATISTICS.
ESTIMATE_ONLY	Calculate job estimates without performing the export.
EXCLUDE	Exclude specific object types, e.g. EXCLUDE=TABLE:EMP.
FILESIZE	Specify the size of each dumpfile in units of bytes.
FLASHBACK_SCN	SCN used to set session snapshot back to.
FLASHBACK_TIME	Time used to get the SCN closest to the specified time.
FULL	Export entire database (N).
HELP	Display Help messages (N).

INCLUDE	Include specific object types, e.g. INCLUDE=TABLE_DATA.
JOB_NAME	Name of export job to create.
LOGFILE	Log file name (export.log).
NETWORK_LINK	Name of remote database link to the source system.
NOLOGFILE	Do not write logfile (N).
PARALLEL	Change the number of active workers for current job.
PARFILE	Specify parameter file.
QUERY	Predicate clause used to export a subset of a table.
SCHEMAS	List of schemas to export (login schema).
STATUS	Frequency (secs) job status is to be monitored where the default (0) will show new status when available.
TABLES	Identifies a list of tables to export - one schema only.
TABLESPACES	Identifies a list of tablespaces to export.
TRANSPORT_FULL_CHECK	Verify storage segments of all tables (N).
TRANSPORT_TABLESPACES	List of tablespaces from which metadata will be unloaded.
VERSION	Version of objects to export where valid keywords are: (COMPATIBLE), LATEST, or any valid database version.

- Oracle generates a system-generated name for the export job unless you specify a name with the JOB_NAME parameter.
- If you specify a name, ensure it does not conflict with a table or view name in your schema because Oracle creates a master table for the export job with the same name as the Data Pump job – this avoids naming conflicts.
- When a job is running, you can execute these commands via Data Pump's interface in interactive mode.

Command	Description
ADD_FILE	Add dumpfile to dumpfile set. ADD_FILE=<dirobj:>dumpfile-name
CONTINUE_CLIENT	Return to logging mode. Job will be re-started if idle.
EXIT_CLIENT	Quit client session and leave job running.
HELP	Summarize interactive commands.
KILL_JOB	Detach and delete job.
PARALLEL	Change the number of active workers for current job. PARALLEL=<number of workers>.
START_JOB	Start/resume current job.
STATUS	Frequency (secs) job status is to be monitored where the default (0) will show new status when available. STATUS=[interval]

STOP_JOB	Orderly shutdown of job execution and exits the client. STOP_JOB=IMMEDIATE performs an immediate shutdown of theData Pump job.

- ➢ Export parameters can be stored to a plain text file and referenced with the PARFILE parameter of the expdp command.
- ➢ Dump files will NOT overwrite previously existing dump files in the same directory.

Import Options

The utility named impdp serves as the interface to Data Pump Import.

- ➢ Like expdp, the impdp utility also has various command-line input parameters to specify characteristics of an import job when one is created.
- ➢ Parameters can also be stored to a parameter file.
- ➢ This table shows the parameters for the impdp utility.

Keyword	Description (Default)
ATTACH	Attach to existing job, e.g. ATTACH [=job name].
CONTENT	Specifies data to load where the valid keywords are: (ALL), DATA_ONLY, and METADATA_ONLY.
DIRECTORY	Directory object to be used for dump, log, and sql files.
DUMPFILE	List of dumpfiles to import from (expdat.dmp), e.g. DUMPFILE=scott1.dmp, scott2.dmp, dmpdir:scott3.dmp.
ESTIMATE	Calculate job estimates where the valid keywords are: (BLOCKS) and STATISTICS.
EXCLUDE	Exclude specific object types, e.g. EXCLUDE=TABLE:EMP.
FLASHBACK_SCN	SCN used to set session snapshot back to.
FLASHBACK_TIME	Time used to get the SCN closest to the specified time.
FULL	Import everything from source (Y).
HELP	Display help messages (N).
INCLUDE	Include specific object types, e.g. INCLUDE=TABLE_DATA.
JOB_NAME	Name of import job to create.
LOGFILE	Log file name (import.log).
NETWORK_LINK	Name of remote database link to the source system.
NOLOGFILE	Do not write logfile.
PARALLEL	Change the number of active workers for current job.
PARFILE	Specify parameter file.

QUERY	Predicate clause used to import a subset of a table.
REMAP_DATAFILE	Redefine datafile references in all DDL statements.
REMAP_SCHEMA	Objects from one schema are loaded into another schema.
REMAP_TABLESPACE	Tablespace object are remapped to another tablespace.
REUSE_DATAFILES	Tablespace will be initialized if it already exists (N).
SCHEMAS	List of schemas to import.
SKIP_UNUSABLE_INDEXES	Skip indexes that were set to the Index Unusable state.
SQLFILE	Write all the SQL DDL to a specified file.
STATUS	Frequency (secs) job status is to be monitored where the default (0) will show new status when available.
STREAMS_CONFIGURATION	Enable the loading of Streams metadata
TABLE_EXISTS_ACTION	Action to take if imported object already exists. Valid keywords: (SKIP), APPEND, REPLACE and TRUNCATE.
TABLES	Identifies a list of tables to import.
TABLESPACES	Identifies a list of tablespaces to import.
TRANSFORM	Metadata transform to apply (Y/N) to specific objects. Valid transform keywords: SEGMENT_ATTRIBUTES and STORAGE. ex. TRANSFORM=SEGMENT_ATTRIBUTES:N:TABLE.
TRANSPORT_DATAFILES	List of datafiles to be imported by transportable mode.
TRANSPORT_FULL_CHECK	Verify storage segments of all tables (N).
TRANSPORT_TABLESPACES	List of tablespaces from which metadata will be loaded. Only valid in NETWORK_LINK mode import operations.
VERSION	Version of objects to export where valid keywords are: (COMPATIBLE), LATEST, or any valid database version. Only valid for NETWORK_LINK and SQLFILE.

Physical Backups (Offline)

A physical backup involves copying the files that comprise the database.
The whole database backup (in Offline mode) is also termed a cold backup. This type of backup will produce a consistent backup.

- The whole database backup when the database is shutdown is consistent as all files have the same SCN.
- The database can be restored from this type of backup without performing recovery; however, this is to a recovery only to the point of the last backup -- not to the point-of-last-committed-transaction.

A cold backup uses operating system command (such as the UNIX and LINUX cp command) to backup while the database is shut down normally (not due to an instance failure).

- This means the shutdown was either: shutdown normal, shutdown immediate, or shutdown transactional.
- If you must execute shutdown abort, then you should restart the database and shutdown normally before taking an offline backup.

Files to backup include:

- Required: All datafiles.
- Required: All control files.
- Required: All online redo log files.
- Optional, but recommended: The init.ora file and server parameter file, and the password file.

- Backups performed using operating system commands while the database is running are NOT valid unless an online backup is being performed.
- Offline backups performed after a database aborts will be inconsistent and may require considerable effort to use for recovery, if they work at all.
- If the instance has crashed, you cannot do a cold backup.

The whole database backup approach can be used with either ARCHIVELOG or NOARCHIVELOG mode.

- If you run in ARCHIVELOG mode, you can take additional recovery steps outlined in these notes to complete a backup to a point-of-last-committed-transaction.
- The Oracle database should be shut down and a full cold backup taken. If this is not possible, develop a hot backup procedure.
- When a full cold backup is taken, archive logs and exports from the time period prior to the backup can be copied to tape and removed from the system.

Obtain a List of Files to Backup

Use SQL*PLUS and query V$DATAFILE to list all datafiles in your database.

```
SELECT name FROM v$datafile;

NAME
--------------------------------------------------------
/u01/student/dbockstd/oradata/USER350system01.dbf
/u01/student/dbockstd/oradata/USER350sysaux01.dbf
/u02/student/dbockstd/oradata/USER350users01.dbf
/u02/student/dbockstd/oradata/USER350data01.dbf
/u03/student/dbockstd/oradata/USER350index01.dbf
/u02/student/dbockstd/oradata/USER350comp_data.dbf
```

```
/u01/student/dbockstd/oradata/USER350undo02.dbf

7 rows selected.
```

Use SQL*PLUS and query the V$PARAMETER view to obtain a list of control files.

```
SELECT value FROM v$parameter
WHERE name = 'control_files';

VALUE
--------------------------------------------------------------------------------
/u01/student/dbockstd/oradata/USER350control01.ctl, /u02/student/dbockstd/oradat
a/USER350control02.ctl, /u03/student/dbockstd/oradata/USER350control03.ctl
```

Directory Structure

A consistent directory structure for datafiles will simplify the backup process.

- Datafiles must be restored to their original location from a backup in order to restart a database without starting in mount mode and specifying where the datafiles are to be relocated.
- Example: This shows datafiles located on three disk drives. Note that the directory structure is consistent.

```
/u01/student/dbockstd/oradata
/u02/student/dbockstd/oradata
/u03/student/dbockstd/oradata
```

The UNIX tar command shown here will backup all files in the oradata directories belonging to dbockstd to a tape drive named /dev/rmt/0hc because the drives are named /u01 through /u03. The –cvf flag creates a new tar saveset.

```
> tar –cvf /dev/rmt/0hc /u0[1-3]/student/dbockstd/oradata
```

Physical Backups (Online)

Online backups are also physical backups, but the database MUST BE running in ARCHIVELOG mode.

- These are also called hot backups (also termed inconsistent backups) because the database is in use – you don't have to shut it down, and this is an important advantage.
- This type of backup can give a read-consistent copy of the database, but will not backup active transactions.
- These are best performed during times of least database activity because online backups use operating system commands to backup physical files – this can affect system performance.
- Online backup involves setting each tablespace into a backup state, backup of the datafiles, and then restoring each tablespace to a normal state.
- Recovery involves using archived redo logs and roll forward to a point in time.
- The following files can be backed up with the database open:
    - All datafiles.
    - All archived redo log files.
    - One control file (via the ALTER DATABASE command).
    - Online backups :
    - Provide full point-in-time recovery.
    - Allow the database to stay open during file system backup.
    -
- Keeps the System Global Area (SGA) of the instance from having to be reset during database backups.

When you tell Oracle to backup an individual datafile or tablespace, Oracle will stop recording checkpoint records in the headers of the online datafiles to be backed up.

- Use the ALTER TABLESPACE BEGIN BACKUP statement to tell Oracle to put a tablespace in hot backup mode.
- If the tablespace is read-only, you can simply backup the online datafiles.
- After completing a hot backup, Oracle advances the file headers to the current database checkpoint after you execute the ALTER TABLESPACE END BACKUP command.
- When tablespaces are backed up, the tablespace is put into an "online backup" mode and the DBWR process writes all blocks to the buffer cache that belong to any file that is part of the tablespace back to disk.
- You must restore the tablespace to normal status once it is backed up or a redo log mismatch will occur and archiving/rollback cannot be successfully accomplished.

Example: A database with 5 tablespaces can have a different tablespace and the control file backed up every night and at the end of a work week, you would have an entire database backup.

The online and archived redo log files are used to make the backup consistent during recovery.

In order to guarantee that you have the redo log files needed to recover an inconsistent backup, you need to issue this SQL statements to force Oracle to switch the current log file and to archive it and all other unarchived log files.

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
ALTER SYSTEM ARCHIVE LOG ALL;
```

If you have log groups, the following SQL statement will archive a specified log group (replace the word integer with the log group number).

```
ALTER SYSTEM ARCHIVE LOG GROUP integer;
```

A hot backup is complex and should be automated with an SQL script. The steps are given below. Also, an automatic backup script should be first tested on a dummy database.

Starting ARCHIVELOG Mode

Ensure that the database is in ARCHIVELOG mode. This series of commands connects as SYS in the SYSDBA role and starts up the dbockstd database in mount mode, then alters the database to start ARCHIVELOG and then opens the database.

```
CONNECT / AS SYSDBA
STARTUP MOUNT
ALTER DATABASE ARCHIVELOG;
ALTER DATABASE OPEN;
```

Performing Online Database Backups

Steps in an online database backup are:

1. Obtain a list of Datafiles to Backup (see the commands given earlier in these notes).
2. Start the hot backup for a tablespace.

```
ALTER TABLESPACE index01 BEGIN BACKUP;
```

3. Backup the datafiles belonging to the tablespace using operating system commands.

```
$ cp /u01/student/dbockstd/oradata/dbockstdINDEX01.dbf
/u03/student/dbockstd/backup/u01/dbockstdINDEX01.dbf
```

4. Indicate the end of the hot backup with the ALTER TABLESPACE command.

```
ALTER TABLESPACE index01 END BACKUP;
```

Datafile backups, which are not as common as tablespace backups, are valid in ARCHIVELOG databases.

The only time a datafile backup is valid for a database in NOARCHIVELOG mode is if every datafile in a tablespace is backed up. You cannot restore the database unless all datafiles are backed up. The datafiles must be read-only or offline-normal.

Datafile Backup Status
A DBA can check the backup status of a datafile by querying the V$BACKUP view.

SELECT file#, status FROM v$backup;

- The term NOT ACTIVE means the datafile is not actively being backed up whereas ACTIVE means the file is being backed up.
- This view is also useful when a database crashes because it shows the backup status of the files at the time of crash.
- This view is NOT useful when the control file in use is a restored backup or a new control file created after the media failure occurred since it will not contain the correct information.
- If you have restored a backup of a file, the V$BACKUP view reflects the backup status of the older version of the file and thus it can contain misleading information.

Backup Archived Logs

After completing an inconsistent backup, backup all archived redo logs that have been produced since the backup began; otherwise, you cannot recover from the backup.

You can delete the original archived logs from the disk.

CONTROL FILE BACKUP

Backup the control file whenever the structure of the database is altered while running in ARCHIVELOG mode.

Examples of structural modifications include the creation of a new tablespace or the movement of a datafile to a new disk drive.

You can backup a control file to a physical file or to a trace file.

Backup a Control File to a Physical File

Use SQLPLUS to generate a binary file.

ALTER DATABASE BACKUP CONTROLFILE TO '/u03/student/dbockstd/backup/dbockstdctrl1.bak' REUSE;

The REUSE clause will overwrite any current backup that exists.

BACKUP TO TRACE FILE.

The TRACE option is used to manage and recover a control file -- it prompts Oracle to write SQL statements to a database trace file rather than generating a physical binary backup file.

ALTER DATABASE BACKUP CONTROLFILE TO TRACE;

- The trace file statements can be used to start the database, recreate the control file, recover, and open the database.
- You can copy the trace file statements to a script file and edit the script to develop a database recovery script if necessary, or to change parameters such as MAXDATAFILES.
- The trace file will be written to the location specified by the USER_DUMP_DEST parameter in the init.ora file.

Complete Media Recovery

Concepts in Media Recovery

- This discussion is based on operating system recovery (not Recovery manager -- RMAN).
- Complete media recovery gives the DBA the option to recover the whole database at one time or to recover individual tablespaces or datafiles one at a time.
- Whichever method you choose (operating system or RMAN), you can recover a database, tablespace, or datafile.
- In order to determine which datafiles need recovery use the fixed view V$RECOVER_FILE which is available by querying a database that is in MOUNTmode.

Closed (Offline) Database Recovery

Media recovery is performed in stages.

SHUTDOWN

- Shut down the database. If the database is open, shut it down with the SHUTDOWN ABORT command.

- Correct the media damage if possible - otherwise consider moving the damaged datafile(s) to existing media if unused disk space is sufficient.

Note: If the hardware problem was temporary and the database is undamaged (disk or controller power failure), start the database and resume normal operations.

STARTUP

- Restore the necessary files. This requires the DBA to determine which datafiles need recovered - remember to query the V$RECOVER_FILE view.
- Permanently damaged files - identify the most recent backups of the damaged files.
- Restore only the damaged datafiles - do not restore any undamaged datafiles or any online redo log files. Use an operating system utility (such as theUNIX cp command to copy) to restore files to their default or not location.
- If you do not have a backup of a specific datafile, you may be able to create an empty replacement file that can be recovered.
- If you can fix the hardware problem (example, replace disk /u02 and format and name the new disk /u02), then restore the datafiles to their original default location.
- If you cannot fix the hardware problem immediately, select an alternate location for the restored datafiles. This will require specifying the new location by using the datafile renaming/relocation procedure specified in the Oracle Administrator's Guide.
- Recover the datafiles.

Connect to Oracle as the DBA with administrator privileges and start a new instance and mount, but do not open the database, e.g. STARTUP MOUNT. Obtain all datafile names by querying the V$DATAFILE view, example:

SELECT name FROM v$datafile;

Ensure datafiles are online.
You may wish to create a script to bring all datafiles online at once or you may decide to alter the database to bring an individual file online. Bring the datafiles online by using one of the following command which Oracle ignores if a datafile is already online, example:

ALTER DATABASE DATAFILE '/u01/student/dbockstd/oradata/dbockstdINDEX01.dbf' ONLINE;
SELECT 'ALTER DATABASE DATAFILE '''||name||''' ONLINE;' FROM v$datafile;

Recover the database with the appropriate command.

RECOVER DATABASE # recovers whole database or use ALTER DATABASE RECOVER

RECOVER TABLESPACE data # recovers specific tablespace

RECOVER DATAFILE '/u10/student/USER310data.dbf'; # recovers specific datafile

APPLY ARCHIVED REDO LOGS
If you do not automate recovery, during execution of the RECOVER command, Oracle will ask you if a specific redo log file is to be applied during recovery - in fact, it will prompt you for the files individually - you simply answer yes or no as appropriate. Alternatively, you can automate recovery and Oracle applies the needed logs automatically by turning on autorecovery, and Oracle will apply the redo log files needed for recovery -- this is the preferred method.

SET AUTORECOVERY ON

· Oracle will notify you when media recovery is finished. Oracle will apply all needed online redo log files and terminate recovery.

OPEN DATABASE
· Use the following command to open the database.

ALTER DATABASE OPEN;

EXAMPLE OF MEDIA RECOVERY

- Suppose that you perform a full backup of all database files by copying them to an offline location on Monday at 1:00 a.m.
- Throughout the rest of the day the database is modified by insertions, deletions, and updates.
- The redo log files switch several times and the database is running in ARCHIVELOG mode.
- At 3:00 p.m., a disk drive containing one tablespace fails.
- Recovery is accomplished by first replacing the disk drive (or using an existing disk drive that has sufficient storage capacity) and then restoring thecomplete database (not just the files for the tablespace that failed) from the last full backup.
- The archived redo logs are next used to recover the database. Oracle uses them automatically when you open the database to the mount stage and issue the ALTER DATABASE RECOVER command. Again, following this alter the database to the open stage: ALTER DATABASE OPEN.
- Following this the database is restarted and the Oracle automatically uses the online redo log files to recover to the point of failure as part of Instance Recovery.

Alternative Media Recovery
If the database file lost is a Temp or Undo tablespace file, you can restore the individual file that failed from the last full backup as described earlier.

Open (Online) Database Recovery

Sometimes a DBA must recover from a media failure while the database remains open.

- This procedure does not apply to the datafiles that constitute the SYSTEM tablespace - damage to this tablespace causes Oracle to shutdown the database.
- In this situation, undamaged datafiles are left online and available for use.
- If DBWR fails to write to a datafile, then Oracle will take the damaged datafiles offline, but not the tablespaces contained in them.
- The stages to open database recovery are discussed below.

1. ENSURE DATABASE IS OPEN.

If the database is not open, startup the database with the STARTUP command.

2. TAKE TABLESPACES OFFLINE.

Take all tablespaces with damaged datafiles offline, example:

```
ALTER TABLESPACE users OFFLINE TEMPORARY;
```

Correct the hardware problem or restore the damaged files to an alternative storage device.

3. RESTORE.

- Restore the most recent backup of files that are permanently damaged by the media failure.
- Do not restore undamaged datafiles, online redo log files, or control files.
- If you have no backup of a specific datafile, use the following command to create an empty replacement file for recovery.

```
ALTER DATABASE CREATE DATAFILE <filename>
```

If you restored a datafile to a new location, use the procedure in the Oracle Administrator's Guide to indicate the new location of the file.

4. RECOVER.

- Connect as a DBA with administrator privileges or as SYS.
- Use the RECOVER TABLESPACE command to start offline tablespace recovery for all damaged datafiles in a tablespace (where the tablespace has more than one datafile, this single command will recover all datafiles for the tablespace).

```
RECOVER TABLESPACE data01 # begins recovery of all datafiles in the data01 tablespace.
```

· At this point Oracle will begin to roll forward by applying all necessary archived redo log files (archived and online) to reconstruct the restored datafiles. You will probably wish to automate this by turning autorecovery on.

```
SET AUTORECOVERY ON
```

- Oracle will continue by applying online redo log files automatically.
- After recovery is complete, bring the offline tablespaces online.

```
ALTER TABLESPACE users ONLINE;
```

Introduction to Oracle Automatic Storage Management

Oracle ASM is a **volume manager** and a **file system** for Oracle database files that supports single-instance Oracle Database and Oracle Real Application Clusters (Oracle RAC) configurations. Oracle ASM is Oracle's recommended storage management solution that provides an alternative to conventional volume managers, file systems, and raw devices.

Oracle ASM uses **disk group**s to store data files; an Oracle ASM disk group is a collection of disks that Oracle ASM manages as a unit. Within a disk group, Oracle ASM exposes a file system interface for Oracle database files. The content of files that are stored in a disk group is evenly distributed to eliminate hot spots and to provide uniform performance across the disks. The performance is comparable to the performance of raw devices.

You can add or remove disks from a disk group while a database continues to access files from the disk group. When you add or remove disks from a disk group, Oracle ASM automatically redistributes the file contents and eliminates the need for downtime when redistributing the content. The Oracle ASM volume manager functionality provides flexible server-based mirroring options. The Oracle ASM normal and high redundancy disk groups enable two-way and three-way mirroring respectively. You can use external redundancy to enable a Redundant Array of Independent Disks (RAID) storage subsystem to perform the mirroring protection function.

Oracle ASM also uses the Oracle Managed Files (OMF) feature to simplify database file management. OMF automatically creates files in designated locations. OMF also names files and removes them while relinquishing space when tablespaces or files are deleted.

Oracle ASM reduces the administrative overhead for managing database storage by consolidating data storage into a small number of disk groups. The smaller number of disk groups consolidates the storage for multiple databases and provides for improved I/O performance.

Oracle ASM files can coexist with other storage management options such as raw disks and third-party file systems. This capability simplifies the integration of Oracle ASM into pre-existing environments.

Understanding Oracle ASM Concepts

About Oracle ASM Instances

An Oracle ASM instance is built on the same technology as an Oracle Database instance. An Oracle ASM instance has a System Global Area (SGA) and background processes that are similar to those of Oracle Database. However, because Oracle ASM performs fewer tasks than a database, an Oracle ASM SGA is much smaller than a database SGA. In addition, Oracle ASM has a minimal performance effect on a server. Oracle ASM instances mount disk groups to make Oracle ASM files available to database instances; Oracle ASM instances do not mount databases.

www.csitportal.com

Oracle ASM is installed in the Oracle Grid Infrastructure home before Oracle Database is installed in a separate Oracle home. Oracle ASM and database instances require shared access to the disks in a disk group. Oracle ASM instances manage the metadata of the disk group and provide file layout information to the database instances.

Oracle ASM metadata is the information that Oracle ASM uses to control a disk group and the metadata resides within the disk group. Oracle ASM metadata includes the following information:

- The disks that belong to a disk group
- The amount of space that is available in a disk group
- The filenames of the files in a disk group
- The location of disk group data file **extent**s
- A redo log that records information about atomically changing metadata blocks
- Oracle ADVM volume information

Oracle ASM instances can be clustered using Oracle Clusterware; there is one Oracle ASM instance for each cluster node. If there are several database instances for different databases on the same node, then the database instances share the same single Oracle ASM instance on that node.

If the Oracle ASM instance on a node fails, then all of the database instances on that node also fail. Unlike a file system driver failure, an Oracle ASM instance failure does not require restarting the operating system. In an Oracle RAC environment, the Oracle ASM and database instances on the surviving nodes automatically recover from an Oracle ASM instance failure on a node.

Figure 1 shows a single node configuration with one Oracle ASM instance and multiple database instances. The Oracle ASM instance manages the metadata and provides space allocation for the Oracle ASM files. When a database instance creates or opens an Oracle ASM file, it communicates those requests to the Oracle ASM instance. In response, the Oracle ASM instance provides file extent map information to the database instance.

In Figure 1, there are two disk groups: one disk group has four disks and the other has two disks. The database can access both disk groups. The configuration in Figure 1 shows multiple database instances, but only one Oracle ASM instance is needed to serve the multiple database instances.

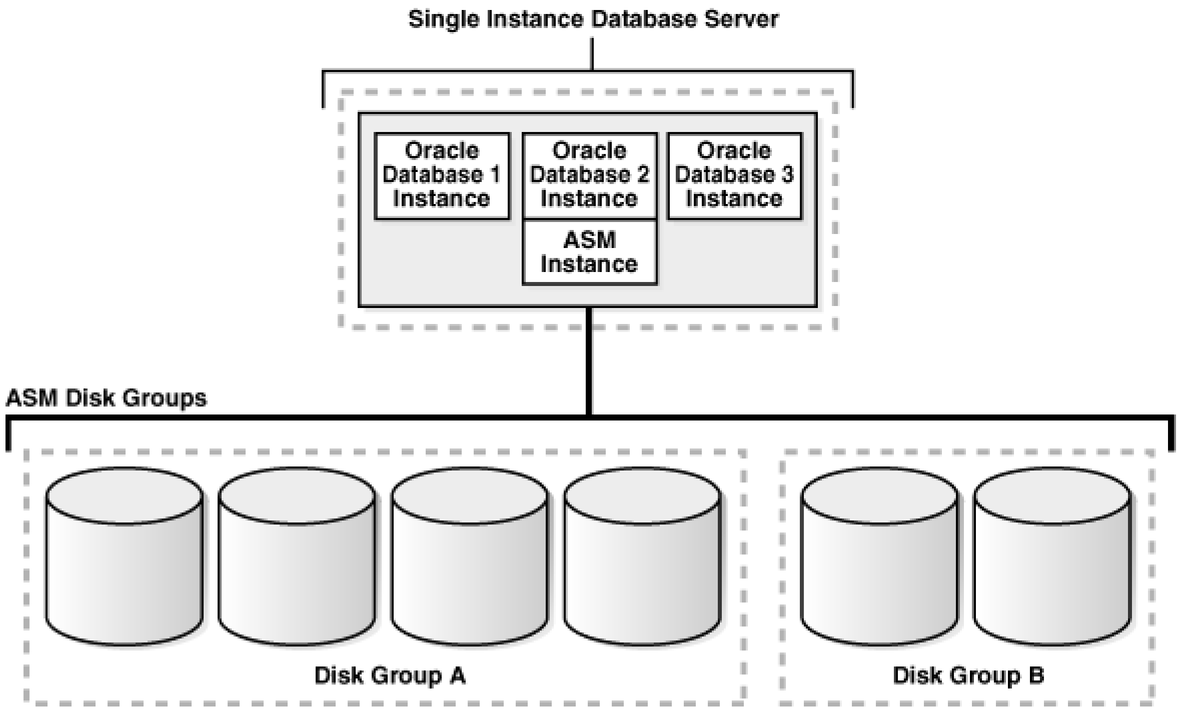

Single Instance Database Server
Oracle Database 1 Instance
Oracle Database 2 Instance
Oracle Database 3 Instance
ASM Instance
ASM Disk Groups
Disk Group A
Disk Group B

Figure 1: Oracle ASM for Single-Instance Oracle Databases

About Oracle ASM Disk Groups

A disk group consists of multiple disks and is the fundamental object that Oracle ASM manages. Each disk group contains the metadata that is required for the management of space in the disk group. Disk group components include disks, files, and allocation units.

Files are allocated from disk groups. Any Oracle ASM file is completely contained within a single disk group. However, a disk group might contain files belonging to several databases and a single database can use files from multiple disk groups. For most installations you need only a small number of disk groups, usually two, and rarely more than three.

About Mirroring and Failure Groups

Mirroring protects data integrity by storing copies of data on multiple disks. When you create a disk group, you specify an Oracle ASM disk group type based on one of the following three redundancy levels:

- **Normal** for 2-way mirroring
- **High** for 3-way mirroring
- **External** to not use Oracle ASM mirroring, such as when you configure hardware RAID for redundancy

The redundancy level controls how many disk failures are tolerated without dismounting the disk group or losing data. The disk group type determines the mirroring levels with which Oracle creates files in a disk group.

Oracle ASM mirroring is more flexible than traditional RAID mirroring. For a disk group specified as **NORMAL** redundancy, you can specify the redundancy level for each file. For example, two files can share the same disk group with one file being mirrored while the other is not.

When Oracle ASM allocates an extent for a mirrored file, Oracle ASM allocates a primary copy and a mirror copy. Oracle ASM chooses the disk on which to store the mirror copy in a different failure group than the primary copy. Failure groups are used to place mirrored copies of data so that each copy is on a disk in a different failure group. The simultaneous failure of all disks in a failure group does not result in data loss.

You define the failure groups for a disk group when you create an Oracle ASM disk group. After a disk group is created, you cannot alter the redundancy level of the disk group. If you omit the failure group specification, then Oracle ASM automatically places each disk into its own failure group, except for disk groups containing disks on Oracle Exadata cells. Normal redundancy disk groups require at least two failure groups. High redundancy disk groups require at least three failure groups. Disk groups with external redundancy do not use failure groups.

About Oracle ASM Disks

Oracle ASM disks are the storage devices that are provisioned to Oracle ASM disk groups. Examples of Oracle ASM disks include:

- A disk or partition from a storage array
- An entire disk or the partitions of a disk
- Logical volumes
- Network-attached files (NFS)

When you add a disk to a disk group, you can assign an Oracle ASM disk name or Oracle ASM assigns the Oracle ASM disk name automatically. This name is different from the path name used by the operating system. In a cluster, a disk may be assigned different operating system device names on different nodes, but the disk has the same Oracle ASM disk name on all of the nodes. In a cluster, an Oracle ASM disk must be accessible from all of the instances that share the disk group.

Oracle ASM spreads the files proportionally across all of the disks in the disk group. This allocation pattern maintains every disk at the same capacity level and ensures that all of the disks in a disk group have the same I/O load. Because Oracle ASM load balances among all of the disks in a disk group, different Oracle ASM disks should not share the same physical drive.

Allocation Units

Every Oracle ASM disk is divided into allocation units (AU). An allocation unit is the fundamental unit of allocation within a disk group. A file extent consists of one or more allocation units. An Oracle ASM file consists of one or more file extents.

When you create a disk group, you can set the Oracle ASM allocation unit size with the **AU_SIZE** disk group attribute. The values can be 1, 2, 4, 8, 16, 32, or 64 MB, depending on the specific disk group compatibility level. Larger AU sizes typically provide performance advantages for data warehouse applications that use large sequential reads.

About Oracle ASM Files

Files that are stored in Oracle ASM disk groups are called Oracle ASM files. Each Oracle ASM file is contained within a single Oracle ASM disk group. Oracle Database communicates with Oracle ASM in terms of files. This is similar to the way Oracle Database uses files on any file system. You can store the various file types in Oracle ASM disk groups, including:

- Control files
- Data files, temporary data files, and data file copies
- SPFILEs
- Online redo logs, archive logs, and Flashback logs
- RMAN backups
- Disaster recovery configurations
- Change tracking bitmaps
- Data Pump dumpsets

Oracle ASM automatically generates Oracle ASM file names as part of file creation and tablespace creation. Oracle ASM file names begin with a plus sign (+) followed by a disk group name. You can specify user-friendly aliases for Oracle ASM files and create a hierarchical directory structure for the aliases.

The following sections describe the Oracle ASM file components:

- Extents
- Oracle ASM Striping
- File Templates

Extents

The contents of Oracle ASM files are stored in a disk group as a set, or collection, of **extent**s that are stored on individual disks within disk groups. Each extent resides on an individual disk. Extents consist of one or more allocation units (AU). To accommodate increasingly larger files, Oracle ASM uses variable size extents.

Variable size extents enable support for larger Oracle ASM data files, reduce SGA memory requirements for very large databases, and improve performance for file create and open operations. The initial extent size equals the disk group allocation unit size and it increases by a factor of 4 or 16 at predefined thresholds.

Oracle ASM Striping

Oracle ASM striping has two primary purposes:

- To balance loads across all of the disks in a disk group
- To reduce I/O latency

Coarse-grained striping provides load balancing for disk groups while fine-grained striping reduces latency for certain file types by spreading the load more widely.

To stripe data, Oracle ASM separates files into stripes and spreads data evenly across all of the disks in a disk group. The fine-grained stripe size always equals 128 KB in any configuration; this provides lower I/O latency for small I/O operations. The coarse-grained stripe size is always equal to the AU size (not the data extent size).

File Templates

Templates are collections of attribute values that are used to specify disk regions, file mirroring, and **striping** attributes for an Oracle ASM file when it is created. When creating a file, you can include a template name and assign desired attributes based on an individual file rather than the file type.

A default template is provided for every Oracle file type, but you can customize templates to meet unique requirements. Each disk group has a default template associated with each file type.

Overview of the RMAN Environment

Recovery Manager (RMAN) is an Oracle Database client that performs backup and recovery tasks on your databases and automates administration of your backup strategies. It greatly simplifies backing up, restoring, and recovering database files.

The RMAN environment consists of the utilities and databases that play a role in backing up your data. At a minimum, the environment for RMAN must include the following components:

A target database

An Oracle database to which RMAN is connected with the TARGET keyword is a target database. A target database is a database on which RMAN is performing backup and recovery operations. RMAN always maintains metadata about its operations on a database in the control file of the database. The RMAN metadata is known as the RMAN repository.

The RMAN client

An Oracle Database executable that interprets commands, directs server sessions to execute those commands, and records its activity in the target database control file.

The RMAN executable is automatically installed with the database and is typically located in the same directory as the other database executables. For example, the RMAN client on Linux is located in $ORACLE_HOME/bin.

Some environments use the following optional components:

A fast recovery area

A disk location in which the database can store and manage files related to backup and recovery. You set the fast recovery area location and size with the DB_R

A media manager

An application required for RMAN to interact with sequential media devices such as tape libraries. A media manager controls these devices during backup and recovery, managing the loading, labeling, and unloading of media. Media management devices are sometimes called SBT (system backup to tape) devices.

A recovery catalog

A separate database schema used to record RMAN activity against one or more target databases. A recovery catalog preserves RMAN repository metadata if the control file is lost, making it much easier to restore and recover following the loss of the control file. The database may overwrite older records in the control file, but RMAN maintains records forever in the catalog unless the records are deleted by the user.

This chapter explains how to use RMAN in the most basic configuration, which is without a recovery catalog or media manager.

Starting RMAN and Connecting to a Database

The RMAN client is started by issuing the rman command at the command prompt of your operating system. RMAN then displays a prompt for your commands as shown in the following example:

```
% rman

RMAN>
```

RMAN connections to a database are specified and authenticated in the same way as SQL*Plus connections to a database. The only difference is that RMAN connections to a target or auxiliary database require the SYSDBA privilege. The AS SYSDBA keywords are implied and cannot be explicitly specified.

Caution:

Good security practice requires that passwords should not be entered in plain text on the command line. You should enter passwords in RMAN only when requested by an RMAN prompt. See Oracle Database Security Guide to learn about password protection.

You can connect to a database with command-line options or by using the CONNECT TARGET command. The following example starts RMAN and then connects to a target database through Oracle Net, AS SYSDBA is not specified because it is implied. RMAN prompts for a password.

```
% rman

RMAN> CONNECT TARGET SYS@prod

target database Password: password

connected to target database: PROD (DBID=39525561)
```

The following variation starts RMAN and then connects to a target database by using operating system authentication:

```
% rman

RMAN> CONNECT TARGET /
```

```
connected to target database: PROD (DBID=39525561)
```

To quit the RMAN client, enter EXIT at the RMAN prompt:

```
RMAN> EXIT
```

Syntax of Common RMAN Command-line Options

```
RMAN

[ TARGET connectStringSpec

| { CATALOG connectStringSpec }

| LOG ['] filename ['] [ APPEND ]

.

.

.

]...


connectStringSpec::=

['] [userid] [/ [password]] [@net_service_name] [']
```

The following example appends the output from an RMAN session to a text file at /tmp/msglog.log

```
% rman TARGET / LOG /tmp/msglog.log APPEND
```

Showing the Default RMAN Configuration

The RMAN backup and recovery environment is preconfigured for each target database. The configuration is persistent and applies to all subsequent operations on this target database, even if you exit and restart RMAN.

RMAN configured settings can specify backup devices, configure a connection to a backup device (known as a channel), policies affecting backup strategy, and others. The default configuration is adequate for most purposes.

To show the current configuration for a database:

Start RMAN and connect to a target database.

Run the SHOW ALL command.

For example, enter the command at the RMAN prompt as follows:

```
RMAN> SHOW ALL;
```

The output lists the CONFIGURE commands to re-create this configuration.

Backing up a Database

Use the BACKUP command to back up files. RMAN backs up data to the configured default device for the type of backup requested. By default, RMAN creates backups on disk. If a fast recovery area is enabled, and if you do not specify the FORMAT parameter, then RMAN creates backups in the recovery area and automatically gives them unique names.

By default, RMAN creates backup sets rather than image copies. A backup set consists of one or more backup pieces, which physical files are written in a format that only RMAN can access. A multiplexed backup set contains the blocks from multiple input files. RMAN can write backup sets to disk or tape.

If you specify BACKUP AS COPY, then RMAN copies each file as an image copy, which is a bit-for-bit copy of a database file created on disk. Image copies are identical to copies created with operating system commands like cp on Linux or COPY on Windows, but are recorded in the RMAN repository and so are usable by RMAN. You can use RMAN to make image copies while the database is open.

Backing Up a Database in ARCHIVELOG Mode

If a database runs in ARCHIVELOG mode, then you can back up the database while it is open. The backup is called an inconsistent backup because redo is required during recovery to bring the database to a consistent state. If you have the archived redo logs needed to recover the backup, open database backups are as effective for data protection as consistent backups.

To back up the database and archived redo logs while the database is open:

Start RMAN and connect to a target database.

Run the BACKUP DATABASE command.

For example, enter the following command at the RMAN prompt to back up the database and all archived redo log files to the default backup device:

```
RMAN> BACKUP DATABASE PLUS ARCHIVELOG;
```

Backing Up a Database in NOARCHIVELOG Mode

If a database runs in NOARCHIVELOG mode, then the only valid database backup is a consistent backup. For the backup to be consistent, the database must be mounted after a consistent shutdown. No recovery is required after restoring the backup.

To make a consistent database backup:

Start RMAN and connect to a target database.

Shut down the database consistently and then mount it.

For example, enter the following commands to guarantee that the database is in a consistent state for a backup:

```
RMAN> SHUTDOWN IMMEDIATE;

RMAN> STARTUP FORCE DBA;

RMAN> SHUTDOWN IMMEDIATE;

RMAN> STARTUP MOUNT;
```

Run the BACKUP DATABASE command.

For example, enter the following command at the RMAN prompt to back up the database to the default backup device:

```
RMAN> BACKUP DATABASE;
```

The following variation of the command creates image copy backups of all data files in the database:

```
RMAN> BACKUP AS COPY DATABASE;
```

Open the database and resume normal operations.

The following command opens the database:

```
RMAN> ALTER DATABASE OPEN;
```

Typical Backup Options

The BACKUP command includes a host of options, parameters, and clauses that control backup output. Table 1 lists some typical backup options.

Table 1 Common Backup Options

Option	Description	Example
FORMAT	Specifies a location and name for backup pieces and copies. You must use substitution variables to generate unique file names. The most common substitution variable is %U, which generates a unique name. Others include %d for the DB_NAME, %t for the backup set time stamp, %s for the backup set number, and %p for the backup piece number.	BACKUP FORMAT 'AL_%d/%t/%s/%p' ARCHIVELOG LIKE '%arc_dest%';
TAG	Specifies a user-defined string as a label for the backup. If you do not specify a tag , then RMAN assigns a default tag with the date and time. Tags are always stored in the RMAN repository in uppercase.	BACKUP TAG 'weekly_full_db_bkup' DATABASE MAXSETSIZE 10M;

Making Incremental Backups

If you specify BACKUP INCREMENTAL, then RMAN creates an incremental backup of a database. Incremental backups capture block-level changes to a database made after a previous incremental backup. Incremental backups are generally smaller and faster to make than full database backups. Recovery with incremental backups is faster than using redo logs alone.

The starting point for an incremental backup strategy is a level 0 incremental backup, which backs up all blocks in the database. An incremental backup at level 0 is identical in content to a full backup, however, unlike a full backup the level 0 backup is considered a part of the incremental backup strategy.

A level 1 incremental backup contains only blocks changed after a previous incremental backup. If no level 0 backup exists in either the current or parent database incarnation when you run a level 1 backup, then RMAN makes a level 0 backup automatically.

Note:

You cannot make incremental backups when a NOARCHIVELOG database is open, although you can make incremental backups when the database is mounted after a consistent shutdown.

A level 1 backup can be a cumulative incremental backup, which includes all blocks changed since the most recent level 0 backup, or a differential incremental backup, which includes only blocks changed since the most recent incremental backup. Incremental backups are differential by default.

When restoring incremental backups, RMAN uses the level 0 backup as the starting point, then updates changed blocks based on level 1 backups where possible to avoid reapplying changes from redo one at a time. Recovering with incremental backups requires no additional effort on your part. If incremental backups are available, then RMAN uses them during recovery.

To make incremental backups of the database:

Start RMAN and connect to a target database.

Run the BACKUP INCREMENTAL command.

The following example creates a level 0 incremental backup to serve as a base for an incremental backup strategy:

BACKUP INCREMENTAL LEVEL 0 DATABASES;

The following example creates a level 1 cumulative incremental backup:

BACKUP INCREMENTAL LEVEL 1 CUMULATIVE DATABASE;

The following example creates a level 1 differential incremental backup:

BACKUP INCREMENTAL LEVEL 1 DATABASE;

Performance Tuning

One of the biggest responsibilities of a DBA is to ensure that the Oracle database is tuned properly. The Oracle RDBMS is highly tunable and allows the database to be monitored and adjusted to increase its performance.

One should do performance tuning for the following reasons:

- The speed of computing might be wasting valuable human time (users waiting for response);
- Enable your system to keep-up with the speed business is conducted; and
- Optimize hardware usage to save money (companies are spending millions on hardware).

As performance bottlenecks are identified during load testing and performance testing, these issues are commonly rectified through a process of performance tuning. Performance tuning can involve configuration changes to hardware, software and network components.

A common bottleneck is the configuration of the application and database servers. Performance tuning can also include tuning SQL queries and tuning an applications underlying code to cater for concurrency

www.csitportal.com

and to improve efficiency. Performance tuning can result in hardware changes being made. This is a last resort; ideally tuning changes will result in a reduction of resource utilization.

Instance Tuning

When considering instance tuning, take care in the initial design of the database to avoid bottlenecks that could lead to performance problems. In addition, you must consider:

- Allocating memory to database structures
- Determining I/O requirements of different parts of the database
- Tuning the operating system for optimal performance of the database

After the database instance has been installed and configured, you must monitor the database as it is running to check for performance-related problems.

Performance Principles

Performance tuning requires a different, although related, method to the initial configuration of a system. Configuring a system involves allocating resources in an ordered manner so that the initial system configuration is functional.

Tuning is driven by identifying the most significant bottleneck and making the appropriate changes to reduce or eliminate the effect of that bottleneck. Usually, tuning is performed reactively, either while the system is in preproduction or after it is live.

Baselines

The most effective way to tune is to have an established performance baseline that you can use for comparison if a performance issue arises. Most database administrators (DBAs) know their system well and can easily identify peak usage periods. For example, the peak periods could be between 10.00am and 12.00pm and also between 1.30pm and 3.00pm. This could include a batch window of 12.00am midnight to 6am.

It is important to identify these peak periods at the site and install a monitoring tool that gathers performance data for those high-load times. Optimally, data gathering should be configured from when the application is in its initial trial phase during the QA cycle. Otherwise, this should be configured when the system is first in production.

Ideally, baseline data gathered should include the following:

- Application statistics (transaction volumes, response time)
- Database statistics
- Operating system statistics
- Disk I/O statistics
- Network statistics

The Symptoms and the Problems

A common pitfall in performance tuning is to mistake the symptoms of a problem for the actual problem itself. It is important to recognize that many performance statistics indicate the symptoms, and that identifying the symptom is not sufficient data to implement a remedy. For example:

Slow physical I/O

Generally, this is caused by poorly-configured disks. However, it could also be caused by a significant amount of unnecessary physical I/O on those disks issued by poorly-tuned SQL.

Latch contention

Rarely is latch contention tunable by reconfiguring the instance. Rather, latch contention usually is resolved through application changes.

Excessive CPU usage

Excessive CPU usage usually means that there is little idle CPU on the system. This could be caused by an inadequately-sized system, by un-tuned SQL statements, or by inefficient application programs.

Proactive Monitoring

Proactive monitoring usually occurs on a regularly scheduled interval, where several performance statistics are examined to identify whether the system behavior and resource usage has changed. Proactive monitoring can also be considered as proactive tuning.

Usually, monitoring does not result in configuration changes to the system, unless the monitoring exposes a serious problem that is developing. In some situations, experienced performance engineers can identify potential problems through statistics alone, although accompanying performance degradation is usual.

Experimenting with or tweaking a system when there is no apparent performance degradation as a proactive action can be a dangerous activity, resulting in unnecessary performance drops. Tweaking a system should be considered reactive tuning, and the steps for reactive tuning should be followed.

Monitoring is usually part of a larger capacity planning exercise, where resource consumption is examined to see changes in the way the application is being used, and the way the application is using the database and host resources.

Bottleneck Elimination

Tuning usually implies fixing a performance problem. However, tuning should be part of the life cycle of an application—through the analysis, design, coding, production, and maintenance stages. Often, the tuning phase is left until the database is in production. At this time, tuning becomes a reactive process, where the most important bottleneck is identified and fixed.

Usually, the purpose for tuning is to reduce resource consumption or to reduce the elapsed time for an operation to complete. Either way, the goal is to improve the effective use of a particular resource. In general, performance problems are caused by the overuse of a particular resource. The overused resource is the bottleneck in the system. There are several distinct phases in identifying the bottleneck and the potential fixes. These are discussed in the sections that follow.

Remember that the different forms of contention are symptoms that can be fixed by making changes in the following places:

- Changes in the application, or the way the application is used
- Changes in Oracle
- Changes in the host hardware configuration

Often, the most effective way of resolving a bottleneck is to change the application.

SQL Tuning

Many application programmers consider SQL a messaging language, because queries are issued and data is returned. However, client tools often generate inefficient SQL statements. Therefore, a good understanding of the database SQL processing engine is necessary for writing optimal SQL. This is especially true for high transaction processing systems.

Typically, SQL statements issued by OLTP applications operate on relatively few rows at a time. If an index can point to the exact rows that are required, then Oracle Database can construct an accurate plan to access those rows efficiently through the shortest possible path. In decision support system (DSS) environments, selectivity is less important, because they often access most of a table's rows. In such situations, full table scans are common, and indexes are not even used.

Query Optimizer and Execution Plans

When a SQL statement is executed on an Oracle database, the query optimizer determines the most efficient execution plan after considering many factors related to the objects referenced and the conditions specified in the query. This determination is an important step in the processing of any SQL statement and can greatly affect execution time.

During the evaluation process, the query optimizer reviews statistics gathered on the system to determine the best data access path and other considerations. You can override the execution plan of the query optimizer with hints inserted in SQL statement.

Introduction to Performance Tuning Features and Tools

Effective data collection and analysis is essential for identifying and correcting performance problems. Oracle Database provides several tools that allow a performance engineer to gather information regarding database performance. In addition to gathering data, Oracle Database provides tools to monitor performance, diagnose problems, and tune applications.

The Oracle Database gathering and monitoring features are mainly automatic, managed by Oracle background processes. To enable automatic statistics collection and automatic performance features, the STATISTICS_LEVEL initialization parameter must be set to TYPICAL or ALL. You can administer and display the output of the gathering and tuning tools with Oracle Enterprise Manager, or with APIs and views. For ease of use and to take advantage of its numerous automated monitoring and diagnostic tools, Oracle Enterprise Manager Database Control is recommended.

Automatic Performance Tuning Features

The Oracle Database automatic performance tuning features include:

Automatic Workload Repository (AWR) collects, processes, and maintains performance statistics for problem detection and self-tuning purposes.

Automatic Database Diagnostic Monitor (ADDM) analyzes the information collected by the AWR for possible performance problems with the Oracle database.

SQL Tuning Advisor allows a quick and efficient technique for optimizing SQL statements without modifying any statements.

Performance Views

The V$ views are the performance information sources used by all Oracle Database performance tuning tools. The V$ views are based on memory structures initialized at instance startup. The memory structures, and the views that represent them, are automatically maintained by Oracle Database for the life of the instance.

Automatic Workload Repository

The Automatic Workload Repository (AWR) collects, processes, and maintains performance statistics for problem detection and self-tuning purposes. This data is both in memory and stored in the database. The gathered data can be displayed in both reports and views.

The statistics collected and processed by AWR include:

- Object statistics that determine both access and usage statistics of database segments
- Time model statistics based on time usage for activities, displayed in the V$SYS_TIME_MODEL and V$SESS_TIME_MODEL views
- Some of the system and session statistics collected in the V$SYSSTAT and V$SESSTAT views
- SQL statements that are producing the highest load on the system, based on criteria such as elapsed time and CPU time
- ASH statistics, representing the history of recent sessions activity

Gathering database statistics using the AWR is enabled by default and is controlled by the STATISTICS_LEVEL initialization parameter. The STATISTICS_LEVEL parameter should be set to the TYPICAL or ALL to enable statistics gathering by the AWR.

Snapshots

Snapshots are sets of historical data for specific time periods that are used for performance comparisons by ADDM. By default, Oracle Database automatically generates snapshots of the performance data once every hour and retains the statistics in the workload repository for 8 days. You can also manually create snapshots, but this is usually not necessary. The data in the snapshot interval is then analyzed by the Automatic Database Diagnostic Monitor (ADDM.

AWR compares the difference between snapshots to determine which SQL statements to capture based on the effect on the system load. This reduces the number of SQL statements that must be captured over time.

Baselines

A baseline contains performance data from a specific time period that is preserved for comparison with other similar workload periods when performance problems occur. The snapshots contained in a baseline are excluded from the automatic AWR purging process and are retained indefinitely.

There are several types of available baselines in Oracle Database:

- Fixed Baselines

www.csitportal.com

- Moving Window Baseline
- Baseline Templates

Fixed Baselines

A fixed baseline corresponds to a fixed, contiguous time period in the past that you specify. Before creating a fixed baseline, carefully consider the time period you choose as a baseline, because the baseline should represent the system operating at an optimal level. In the future, you can compare the baseline with other baselines or snapshots captured during periods of poor performance to analyze performance degradation over time.

Moving Window Baseline

A moving window baseline corresponds to all AWR data that exists within the AWR retention period. This is useful when using adaptive thresholds because the database can use AWR data in the entire AWR retention period to compute metric threshold values.

Oracle Database automatically maintains a system-defined moving window baseline. The default window size for the system-defined moving window baseline is the current AWR retention period, which by default is 8 days.

Baseline Templates

You can also create baselines for a contiguous time period in the future using baseline templates. There are two types of baseline templates: single and repeating.

You can use a single baseline template to create a baseline for a single contiguous time period in the future. This technique is useful if you know beforehand of a time period that you intend to capture in the future. For example, you may want to capture the AWR data during a system test that is scheduled for the upcoming weekend. In this case, you can create a single baseline template to automatically capture the time period when the test occurs.

You can use a repeating baseline template to create and drop baselines based on a repeating time schedule. This is useful if you want Oracle Database to automatically capture a contiguous time period on an ongoing basis. For example, you may want to capture the AWR data during every Monday morning for a month. In this case, you can create a repeating baseline template to automatically create baselines on a repeating schedule for every Monday, and automatically remove older baselines after a specified expiration interval, such as one month.

Adaptive Thresholds

Adaptive thresholds enable you to monitor and detect performance issues while minimizing administrative overhead. Adaptive thresholds can automatically set warning and critical alert thresholds for some system metrics using statistics derived from metric values captured in the moving window baseline. The statistics for these thresholds are recomputed weekly and might result in new thresholds as system performance evolves over time. In addition to recalculating

www.csitportal.com

thresholds weekly, adaptive thresholds might compute different thresholds values for different times of the day or week based on periodic workload patterns.

Space Consumption

The space consumed by the AWR is determined by several factors:

- Number of active sessions in the system at any given time
- Snapshot interval

  The snapshot interval determines the frequency at which snapshots are captured. A smaller snapshot interval increases the frequency, which increases the volume of data collected by the AWR.

- Historical data retention period

  The retention period determines how long this data is retained before being purged. A longer retention period increases the space consumed by the AWR.

By default, snapshots are captured once every hour and are retained in the database for 8 days. With these default settings, a typical system with an average of 10 concurrent active sessions can require approximately 200 to 300 MB of space for its AWR data. It is possible to change the default values for both snapshot interval and retention period. See "Modifying Snapshot Settings" to learn how to modify AWR settings.

The AWR space consumption can be reduced by the increasing the snapshot interval and reducing the retention period. When reducing the retention period, note that several Oracle Database self-managing features depend on AWR data for proper functioning. Not having enough data can affect the validity and accuracy of these components and features, including:

- Automatic Database Diagnostic Monitor
- SQL Tuning Advisor
- Undo Advisor
- Segment Advisor

If possible, Oracle recommends that you set the AWR retention period large enough to capture at least one complete workload cycle. If your system experiences weekly workload cycles, such as OLTP workload during weekdays and batch jobs during the weekend, you do not need to change the default AWR retention period of 8 days. However if your system is subjected to a monthly peak load during month end book closing, you may have to set the retention period to one month.

Under exceptional circumstances, you can turn off automatic snapshot collection by setting the snapshot interval to 0. Under this condition, the automatic collection of the workload and statistical data is stopped and much of the Oracle Database self-management functionality is not operational. In addition, you cannot manually create snapshots. For this reason, Oracle strongly recommends that you do not turn off automatic snapshot collection.

Managing the Automatic Workload Repository

Managing Snapshots

By default, Oracle Database generates snapshots once every hour, and retains the statistics in the workload repository for 8 days. When necessary, you can use DBMS_WORKLOAD_REPOSITORY procedures to manually create, drop, and modify the snapshots. To invoke these procedures, a user must be granted the DBA role.

The primary interface for managing snapshots is Oracle Enterprise Manager. If Oracle Enterprise Manager is unavailable, you can manage snapshots using the DBMS_WORKLOAD_REPOSITORY package, as described in the following sections:

- Creating Snapshots
- Dropping Snapshots
- Modifying Snapshot Settings

Creating Snapshots

You can manually create snapshots with the CREATE_SNAPSHOT procedure to capture statistics at times different than those of the automatically generated snapshots. For example:

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
END;
/
```

In this example, a snapshot for the instance is created immediately with the flush level specified to the default flush level of TYPICAL. You can view this snapshot in the DBA_HIST_SNAPSHOT view.

Dropping Snapshots

You can drop a range of snapshots using the DROP_SNAPSHOT_RANGE procedure. To view a list of the snapshot IDs along with database IDs, check the DBA_HIST_SNAPSHOT view. For example, you can drop the following range of snapshots:

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE (low_snap_id => 22,
                         high_snap_id => 32, dbid => 3310949047);
END;
/
```

In the example, the range of snapshot IDs to drop is specified from 22 to 32. The optional database identifier is 3310949047. If you do not specify a value for dbid, the local database identifier is used as the default value.

Active Session History data (ASH) that belongs to the time period specified by the snapshot range is also purged when the DROP_SNAPSHOT_RANGE procedure is called.

Modifying Snapshot Settings

You can adjust the interval, retention, and captured Top SQL of snapshot generation for a specified database ID, but note that this can affect the precision of the Oracle Database diagnostic tools.

The INTERVAL setting affects how often the database automatically generates snapshots. The RETENTION setting affects how long the database stores snapshots in the workload repository. The TOPNSQL setting affects the number of Top SQL to flush for each SQL criteria (Elapsed Time, CPU Time, Parse Calls, sharable Memory, and Version Count). The value for this setting is not affected by the statistics/flush level and will override the system default behavior for the AWR SQL collection. It is possible to set the value for this setting to MAXIMUM to capture the complete set of SQL in the shared SQL area, though by doing so (or by setting the value to a very high number) may lead to possible space and performance issues because there will more data to collect and store. To adjust the settings, use the MODIFY_SNAPSHOT_SETTINGS procedure. For example:

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS( retention => 43200,
              interval => 30, topnsql => 100, dbid => 3310949047);
END;
/
```

In this example, the retention period is specified as 43200 minutes (30 days), the interval between each snapshot is specified as 30 minutes, and the number of Top SQL to flush for each SQL criteria as 100. If NULL is specified, the existing value is preserved. The optional database identifier is 3310949047. If you do not specify a value for dbid, the local database identifier is used as the default value. You can check the current settings for your database instance with the DBA_HIST_WR_CONTROL view.

Managing Baselines

This section describes how to manage baselines. The primary interface for managing baselines is Oracle Enterprise Manager. Whenever possible, you should manage baselines using Oracle Enterprise Manager, as described in Oracle Database 2 Day + Performance Tuning Guide. If Oracle Enterprise Manager is unavailable, you can manage baselines using the DBMS_WORKLOAD_REPOSITORY package, as described in the following sections:

- Creating a Baseline
- Dropping a Baseline
- Renaming a Baseline
- Displaying Baseline Metrics
- Modifying the Window Size of the Default Moving Window Baseline

Creating a Baseline

This section describes how to create a baseline using an existing range of snapshots.

To create a baseline:

1. Review the existing snapshots in the DBA_HIST_SNAPSHOT view to determine the range of snapshots to use.
2. Use the CREATE_BASELINE procedure to create a baseline using the desired range of snapshots:

```
BEGIN
    DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE (start_snap_id => 270,
                   end_snap_id => 280, baseline_name => 'peak baseline',
                   dbid => 3310949047, expiration => 30);
END;
/
```

In this example, 270 is the start snapshot sequence number and 280 is the end snapshot sequence. The name of baseline is peak baseline. The optional database identifier is 3310949047. If you do not specify a value for dbid, then the local database identifier is used as the default value. The optional expiration parameter is set to 30, so the baseline will expire and be dropped automatically after 30 days. If you do not specify a value for expiration, the baseline will never expire.

The system automatically assign a unique baseline ID to the new baseline when the baseline is created. The baseline ID and database identifier are displayed in the DBA_HIST_BASELINE view.

Dropping a Baseline

This section describes how to drop an existing baseline. Periodically, you may want to drop a baseline that is no longer used to conserve disk space. The snapshots associated with a baseline are retained indefinitely until you explicitly drop the baseline or the baseline has expired.

To drop a baseline:

1. Review the existing baselines in the DBA_HIST_BASELINE view to determine the baseline to drop.
2. Use the DROP_BASELINE procedure to drop the desired baseline:

```
BEGIN
  DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE (baseline_name => 'peak baseline',
                 cascade => FALSE, dbid => 3310949047);
END;
/
```

In the example, the name of baseline is peak baseline. The cascade parameter is set to FALSE, which specifies that only the baseline is dropped. Setting this parameter to TRUE specifies that the drop operation will also remove the snapshots associated with the baseline. The optional dbid parameter specifies the database identifier, which in this example is

www.csitportal.com

3310949047. If you do not specify a value for dbid, then the local database identifier is used as the default value.

Renaming a Baseline

This section describes how to rename a baseline.

To rename a baseline:

1. Review the existing baselines in the DBA_HIST_BASELINE view to determine the baseline to rename.
2. Use the RENAME_BASELINE procedure to rename the desired baseline:
3. BEGIN
4.   DBMS_WORKLOAD_REPOSITORY.RENAME_BASELINE (
5.   old_baseline_name => 'peak baseline',
6.   new_baseline_name => 'peak mondays',
7.   dbid => 3310949047);
8. END;
9. /

   In this example, the name of the baseline is renamed from peak baseline, as specified by the old_baseline_name parameter, to peak mondays, as specified by the new_baseline_name parameter. The optional dbid parameter specifies the database identifier, which in this example is 3310949047. If you do not specify a value for dbid, then the local DBID is the default value.

Displaying Baseline Metrics

This section describes how to display metric threshold settings during the time period captured in a baseline. When used with adaptive thresholds, a baseline contains AWR data that the database can use to compute metric threshold values. The SELECT_BASELINE_METRICS function enables you to display the summary statistics for metric values in a baseline period.

To display metric information in a baseline:

1. Review the existing baselines in the DBA_HIST_BASELINE view to determine the baseline for which you want to display metric information.
2. Use the SELECT_BASELINE_METRICS function to display the metric information for the desired baseline:
3. BEGIN
4.   DBMS_WORKLOAD_REPOSITORY.SELECT_BASELINE_METRICS (
5.   baseline_name => 'peak baseline',
6.   dbid => 3310949047,
7.   instance_num => '1');
8. END;
9. /

   In this example, the name of baseline is peak baseline. The optional dbid parameter specifies the database identifier, which in this example is 3310949047. If you do not specify a value for dbid, then the local database identifier is used as the default value. The optional instance_num parameter specifies the instance number, which in this example is 1. If you do not specify a value for instance_num, then the local instance is used as the default value.

Modifying the Window Size of the Default Moving Window Baseline

This section describes how to modify the window size of the default moving window baseline. For information about the default moving window baseline, see "Moving Window Baseline".

To resize the default moving window baseline, use the MODIFY_BASELINE_WINDOW_SIZE procedure:

```
BEGIN
   DBMS_WORKLOAD_REPOSITORY.MODIFY_BASELINE_WINDOW_SIZE (
              window_size => 30,
              dbid => 3310949047);
END;
/
```

The window_size parameter is used to specify the new window size, in number of days, for the default moving window size. In this example, the window_size parameter is set to 30. The window size must be set to a value that is equal to or less than the value of the AWR retention setting. To set a window size that is greater than the current AWR retention period, you must first increase the value of the retention parameter, as described in "Modifying Snapshot Settings".

In this example, the optional dbid parameter specifies the database identifier is 3310949047. If you do not specify a value for dbid, then the local database identifier is used as the default value.

Managing Baseline Templates

This section describes how to manage baseline templates. You can automatically create baselines to capture specified time periods in the future using baseline templates. For information about baseline templates, see "Baseline Templates".

The primary interface for managing baseline templates is Oracle Enterprise Manager. Whenever possible, you should manage baseline templates using Oracle Enterprise Manager, as described in Oracle Database 2 Day + Performance Tuning Guide. If Oracle Enterprise Manager is unavailable, you can manage baseline templates using the DBMS_WORKLOAD_REPOSITORY package, as described in the following sections:

- Creating a Single Baseline Template
- Creating a Repeating Baseline Template
- Dropping a Baseline Template

See Also:

Oracle Database PL/SQL Packages and Types Reference for detailed information on the DBMS_WORKLOAD_REPOSITORY package

Creating a Single Baseline Template

This section describes how to create a single baseline template. You can use a single baseline template to create a baseline during a single, fixed time interval in the future. For example, you can create a single baseline template to generate a baseline that is captured on April 2, 2009 from 5:00 p.m. to 8:00 p.m.

To create a single baseline template, use the CREATE_BASELINE_TEMPLATE procedure:

```
BEGIN
   DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (
               start_time => '2009-04-02 17:00:00 PST',
               end_time => '2009-04-02 20:00:00 PST',
               baseline_name => 'baseline_090402',
               template_name => 'template_090402', expiration => 30,
               dbid => 3310949047);
END;
/
```

The start_time parameter specifies the start time for the baseline to be created. The end_time parameter specifies the end time for the baseline to be created. The baseline_name parameter specifies the name of the baseline to be created. The template_name parameter specifies the name of the baseline template. The optional expiration parameter specifies the expiration, in number of days, for the baseline. If unspecified, then the baseline never expires. The optional dbid parameter specifies the database identifier. If unspecified, then the local database identifier is used as the default value.

In this example, a baseline template named template_090402 is created that will generate a baseline named baseline_090402 for the time period from 5:00 p.m. to 8:00 p.m. on April 2, 2009 on the database with a database ID of 3310949047. The baseline will expire after 30 days.

Creating a Repeating Baseline Template

This section describes how to create a repeating baseline template. A repeating baseline template can be used to automatically create baselines that repeat during a particular time interval over a specific period in the future. For example, you can create a repeating baseline template to generate a baseline that repeats every Monday from 5:00 p.m. to 8:00 p.m. for the year 2009.

To create a repeating baseline template, use the CREATE_BASELINE_TEMPLATE procedure:

```
BEGIN
   DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE_TEMPLATE (
               day_of_week => 'monday', hour_in_day => 17,
               duration => 3, expiration => 30,
               start_time => '2009-04-02 17:00:00 PST',
               end_time => '2009-12-31 20:00:00 PST',
               baseline_name_prefix => 'baseline_2009_mondays_',
               template_name => 'template_2009_mondays',
               dbid => 3310949047);
END;
/
```

The day_of_week parameter specifies the day of the week on which the baseline will repeat. The hour_in_day parameter specifies the hour in the day when the baseline will start. The duration parameter specifies the duration, in number of hours, that the baseline will last. The expiration parameter specifies the number of days to retain each created baseline. If set to NULL, then the baselines never expires. The start_time parameter specifies the start time for the baseline to be created. The end_time parameter specifies the end time for the baseline to be created. The baseline_name_prefix parameter specifies the name of the baseline prefix that will be appended to the data information when the baseline is created. The template_name parameter specifies the name of the baseline template. The optional dbid parameter specifies the database identifier. If unspecified, then the local database identifier is used as the default value.

In this example, a baseline template named template_2009_mondays is created that will generate a baseline on every Monday from 5:00 p.m. to 8:00 p.m. beginning on April 2, 2009 at 5:00 p.m. and ending on December 31, 2009 at 8:00 p.m. on the database with a database ID of 3310949047. Each of the baselines will be created with a baseline name with the prefix baseline_2009_mondays_ and will expire after 30 days.

Dropping a Baseline Template

This section describes how to drop an existing baseline template. Periodically, you may want to remove baselines templates that are no longer used to conserve disk space.

To drop a baseline template:

1. Review the existing baselines in the DBA_HIST_BASELINE_TEMPLATE view to determine the baseline template you want to drop.
2. Use the DROP_BASELINE_TEMPLATE procedure to drop the desired baseline template:
3. BEGIN
4. DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE_TEMPLATE (
5. template_name => 'template_2009_mondays',
6. dbid => 3310949047);
7. END;
8. /

   The template_name parameter specifies the name of the baseline template that will be dropped. In the example, the name of baseline template that will be dropped is template_2009_mondays. The optional dbid parameter specifies the database identifier, which in this example is 3310949047. If you do not specify a value for dbid, then the local database identifier is used as the default value.

Generating Automatic Workload Repository Reports

www.csitportal.com

An AWR report shows data captured between two snapshots (or two points in time). The AWR reports are divided into multiple sections. The HTML report includes links that can be used to navigate quickly between sections. The content of the report contains the workload profile of the system for the selected range of snapshots.

The primary interface for generating AWR reports is Oracle Enterprise Manager. Whenever possible, you should generate AWR reports using Oracle Enterprise Manager, as described in Oracle Database 2 Day + Performance Tuning Guide. If Oracle Enterprise Manager is unavailable, you can generate AWR reports by running SQL scripts, as described in the following sections:

- Generating an AWR Report
- Generating an Oracle RAC AWR Report
- Generating an AWR Report on a Specific Database Instance
- Generating an Oracle RAC AWR Report on Specific Database Instances
- Generating an AWR Report for a SQL Statement
- Generating an AWR Report for a SQL Statement on a Specific Database Instance

To run these scripts, you must be granted the DBA role.

Note:

If you run a report on a database that does not have any workload activity during the specified range of snapshots, calculated percentages for some report statistics can be less than 0 or greater than 100. This result simply means that there is no meaningful value for the statistic.

Generating an AWR Report

The awrrpt.sql SQL script generates an HTML or text report that displays statistics for a range of snapshot IDs.

To generate an AWR report:

1. At the SQL prompt, enter:
2. @$ORACLE_HOME/rdbms/admin/awrrpt.sql
3. Specify whether you want an HTML or a text report:
4. Enter value for report_type: text

   In this example, a text report is chosen.

5. Specify the number of days for which you want to list snapshot IDs.
6. Enter value for num_days: 2

   A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

7. Specify a beginning and ending snapshot ID for the workload repository report:

8. Enter value for begin_snap: 150
9. Enter value for end_snap: 160

   In this example, the snapshot with a snapshot ID of 150 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 160 is selected as the ending snapshot.

10. Enter a report name, or accept the default report name:
11. Enter value for report_name:
12. Using the report name awrrpt_1_150_160

    In this example, the default name is accepted and an AWR report named awrrpt_1_150_160 is generated.

Generating an AWR Report on a Specific Database Instance

The awrrpti.sql SQL script generates an HTML or text report that displays statistics for a range of snapshot IDs using a specific database and instance. This script enables you to specify a database identifier and instance for which the AWR report will be generated.

To generate an AWR report on a specific database instance:

1. At the SQL prompt, enter:
2. @$ORACLE_HOME/rdbms/admin/awrrpti.sql
3. Specify whether you want an HTML or a text report:
4. Enter value for report_type: text

   In this example, a text report is chosen.

   A list of available database identifiers and instance numbers are displayed:

```
Instances in this Workload Repository schema
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   DB Id    Inst Num DB Name      Instance     Host
----------- -------- ------------ ------------ ------------
 3309173529        1 MAIN         main         examp1690
 3309173529        1 TINT251      tint251      samp251
```

5. Enter the values for the database identifier (dbid) and instance number (inst_num):
6. Enter value for dbid: 3309173529
7. Using 3309173529 for database Id
8. Enter value for inst_num: 1
9. Specify the number of days for which you want to list snapshot IDs.
10. Enter value for num_days: 2

    A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the last 2 days are displayed.

11. Specify a beginning and ending snapshot ID for the workload repository report:
12. Enter value for begin_snap: 150

13. Enter value for end_snap: 160

    In this example, the snapshot with a snapshot ID of 150 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 160 is selected as the ending snapshot.

14. Enter a report name, or accept the default report name:
15. Enter value for report_name:
16. Using the report name awrrpt_1_150_160

    In this example, the default name is accepted and an AWR report named awrrpt_1_150_160 is generated on the database instance with a database ID value of 3309173529.

Generating an AWR Report for a SQL Statement

The awrsqrpt.sql SQL script generates an HTML or text report that displays statistics of a particular SQL statement for a range of snapshot IDs. Run this report to inspect or debug the performance of a SQL statement.

To generate an AWR report for a particular SQL statement:

1. At the SQL prompt, enter:
2. @$ORACLE_HOME/rdbms/admin/awrsqrpt.sql
3. Specify whether you want an HTML or a text report:
4. Enter value for report_type: html

    In this example, an HTML report is chosen.

5. Specify the number of days for which you want to list snapshot IDs.
6. Enter value for num_days: 1

    A list of existing snapshots for the specified time range is displayed. In this example, snapshots captured in the previous day are displayed.

7. Specify a beginning and ending snapshot ID for the workload repository report:
8. Enter value for begin_snap: 146
9. Enter value for end_snap: 147

    In this example, the snapshot with a snapshot ID of 146 is selected as the beginning snapshot, and the snapshot with a snapshot ID of 147 is selected as the ending snapshot.

10. Specify the SQL ID of a particular SQL statement to display statistics:
11. Enter value for sql_id: 2b064ybzkwf1y

    In this example, the SQL statement with a SQL ID of 2b064ybzkwf1y is selected.

12. Enter a report name, or accept the default report name:
13. Enter value for report_name:
14. Using the report name awrrpt_1_146_147.html

In this example, the default name is accepted and an AWR report named awrrpt_1_146_147 is generated.

Automatic Database Diagnostic Monitor

When problems occur with a system, it is important to perform accurate and timely diagnosis of the problem before making any changes to a system. Oftentimes, a database administrator (DBA) simply looks at the symptoms and immediately starts changing the system to fix those symptoms. However, an accurate diagnosis of the actual problem in the initial stage significantly increases the probability of success in resolving the problem.

With Oracle Database, the statistical data needed for accurate diagnosis of a problem is stored in the Automatic Workload Repository (AWR). The Automatic Database Diagnostic Monitor (ADDM):

- Analyzes the AWR data on a regular basis
- Diagnoses the root causes of performance problems
- Provides recommendations for correcting any problems
- Identifies non-problem areas of the system

Because AWR is a repository of historical performance data, ADDM can analyze performance issues after the event, often saving time and resources in reproducing a problem. In most cases, ADDM output should be the first place that a DBA looks when notified of a performance problem. ADDM provides the following benefits:

- Automatic performance diagnostic report every hour by default
- Problem diagnosis based on decades of tuning expertise
- Time-based quantification of problem impacts and recommendation benefits
- Identification of root cause, not symptoms
- Recommendations for treating the root causes of problems
- Identification of non-problem areas of the system
- Minimal overhead to the system during the diagnostic process

It is important to realize that tuning is an iterative process, and fixing one problem can cause the bottleneck to shift to another part of the system. Even with the benefit of ADDM analysis, it can take multiple tuning cycles to reach acceptable system performance. ADDM benefits apply beyond production systems; on development and test systems, ADDM can provide an early warning of performance issues.

ADDM Analysis

An ADDM analysis can be performed on a pair of AWR snapshots and a set of instances from the same database. The pair of AWR snapshots define the time period for analysis, and the set of instances define the target for analysis.

If you are using Oracle Real Application Clusters (Oracle RAC), ADDM has three analysis modes:

- Database
  In Database mode, ADDM analyzes all instances of the database.
- Instance
  In Instance mode, ADDM analyzes a particular instance of the database.
- Partial
  In Partial mode, ADDM analyzes a subset of all database instances.

An ADDM analysis is performed each time an AWR snapshot is taken and the results are saved in the database. The time period analyzed by ADDM is defined by the last two snapshots (the last hour by default). ADDM will always analyze the specified instance in Instance mode. For non-Oracle RAC or single instance environments, the analysis performed in the Instance mode is the same as a database-wide analysis. After an ADDM completes its analysis, you can view the results using Oracle Enterprise Manager, or by viewing a report in a SQL*Plus session.

ADDM analysis is performed top down, first identifying symptoms, and then refining them to reach the root causes of performance problems. The goal of the analysis is to reduce a single throughput metric called DB time. DB time is the cumulative time spent by the database in processing user requests. It includes wait time and CPU time of all non-idle user sessions. DB time is displayed in the V$SESS_TIME_MODEL and V$SYS_TIME_MODEL views.

By reducing DB time, the database is able to support more user requests using the same resources, which increases throughput. The problems reported by ADDM are sorted by the amount of DB time they are responsible for. System areas that are not responsible for a significant portion of DB time are reported as non-problem areas.

The types of problems that ADDM considers include the following:

- CPU bottlenecks - Is the system CPU bound by Oracle Database or some other application?
- Undersized Memory Structures - Are the Oracle Database memory structures, such as the SGA, PGA, and buffer cache, adequately sized?
- I/O capacity issues - Is the I/O subsystem performing as expected?
- High load SQL statements - Are there any SQL statements which are consuming excessive system resources?
- High load PL/SQL execution and compilation, and high-load Java usage
- Oracle RAC specific issues - What are the global cache hot blocks and objects; are there any interconnect latency issues?
- Sub-optimal use of Oracle Database by the application - Are there problems with poor connection management, excessive parsing, or application level lock contention?
- Database configuration issues - Is there evidence of incorrect sizing of log files, archiving issues, excessive checkpoints, or sub-optimal parameter settings?
- Concurrency issues - Are there buffer busy problems?
- Hot objects and top SQL for various problem areas

www.csitportal.com

ADDM also documents the non-problem areas of the system. For example, wait event classes that are not significantly impacting the performance of the system are identified and removed from the tuning consideration at an early stage, saving time and effort that would be spent on items that do not impact overall system performance.

ADDM Analysis Results

In addition to problem diagnostics, ADDM recommends possible solutions. ADDM analysis results are represented as a set of findings. See Example 1 for an example of ADDM analysis result. Each ADDM finding can belong to one of the following types:

- Problem findings describe the root cause of a database performance problem.
- Symptom findings contain information that often lead to one or more problem findings.
- Information findings are used for reporting information that are relevant to understanding the performance of the database, but do not constitute a performance problem (such as non-problem areas of the database and the activity of automatic database maintenance).
- Warning findings contain information about problems that may affect the completeness or accuracy of the ADDM analysis (such as missing data in the AWR).

Each problem finding is quantified by an impact that is an estimate of the portion of DB time caused by the finding's performance issue. A problem finding can be associated with a list of recommendations for reducing the impact of the performance problem. The types of recommendations include:

- Hardware changes: adding CPUs or changing the I/O subsystem configuration
- Database configuration: changing initialization parameter settings
- Schema changes: hash partitioning a table or index, or using automatic segment-space management (ASSM)
- Application changes: using the cache option for sequences or using bind variables
- Using other advisors: running SQL Tuning Advisor on high-load SQL or running the Segment Advisor on hot objects

Reviewing ADDM Analysis Results: Example

Consider the following section of an ADDM report in Example 1.

Example 6-1 Example ADDM Report

```
FINDING 1: 31% impact (7798 seconds)
-----------------------------------
SQL statements were not shared due to the usage of literals. This resulted in
additional hard parses which were consuming significant database time.

RECOMMENDATION 1: Application Analysis, 31% benefit (7798 seconds)
  ACTION: Investigate application logic for possible use of bind variables
    instead of literals. Alternatively, you may set the parameter
    "cursor_sharing" to "force".
  RATIONALE: SQL statements with PLAN_HASH_VALUE 3106087033 were found to be
```

www.csitportal.com

using literals. Look in V$SQL for examples of such SQL statements.

In Example 1, the finding points to a particular root cause, the usage of literals in SQL statements, which is estimated to have an impact of about 31% of total DB time in the analysis period.

The finding has a recommendation associated with it, composed of one action and one rationale. The action specifies a solution to the problem found and is estimated to have a maximum benefit of up to 31% DB time in the analysis period. Note that the benefit is given as a portion of the total DB time and not as a portion of the finding's impact. The rationale provides additional information on tracking potential SQL statements that were using literals and causing this performance issue. Using the specified plan hash value of SQL statements that could be a problem, a DBA could quickly examine a few sample statements.

Setting Up ADDM

Automatic database diagnostic monitoring is enabled by default and is controlled by the CONTROL_MANAGEMENT_PACK_ACCESS and the STATISTICS_LEVEL initialization parameters.

The CONTROL_MANAGEMENT_PACK_ACCESS parameter should be set to DIAGNOSTIC or DIAGNOSTIC+TUNING to enable automatic database diagnostic monitoring. The default setting is DIAGNOSTIC+TUNING. Setting CONTROL_MANAGEMENT_PACK_ACCESS to NONE disables ADDM.

The STATISTICS_LEVEL parameter should be set to the TYPICAL or ALL to enable automatic database diagnostic monitoring. The default setting is TYPICAL. Setting STATISTICS_LEVEL to BASIC disables many Oracle Database features, including ADDM, and is strongly discouraged.

See Also:

Oracle Database Reference for information about the CONTROL_MANAGEMENT_PACK_ACCESS and STATISTICS_LEVEL initialization parameters

ADDM analysis of I/O performance partially depends on a single argument, DBIO_EXPECTED, that describes the expected performance of the I/O subsystem. The value of DBIO_EXPECTED is the average time it takes to read a single database block in microseconds. Oracle Database uses the default value of 10 milliseconds, which is an appropriate value for most modern hard drives. If your hardware is significantly different, such as very old hardware or very fast RAM disks, consider using a different value.

To determine the correct setting for DBIO_EXPECTED parameter:

1. Measure the average read time of a single database block read for your hardware. Note that this measurement is for random I/O, which includes seek time if you use standard hard drives. Typical values for hard drives are between 5000 and 20000 microseconds.

www.csitportal.com

2. Set the value one time for all subsequent ADDM executions. For example, if the measured value if 8000 microseconds, you should execute the following command as SYS user:
3. EXECUTE DBMS_ADVISOR.SET_DEFAULT_TASK_PARAMETER(
4. 'ADDM', 'DBIO_EXPECTED', 8000);

Diagnosing Database Performance Problems with ADDM

To diagnose database performance problems, first review the ADDM analysis results that are automatically created each time an AWR snapshot is taken. If a different analysis is required (such as a longer analysis period, using a different DBIO_EXPECTED setting, or changing the analysis mode), you can run ADDM manually as described in this section.

ADDM can analyze any two AWR snapshots (on the same database), as long as both snapshots are still stored in the AWR (have not been purged). ADDM can only analyze instances that are started before the beginning snapshot and remain running until the ending snapshot. Additionally, ADDM will not analyze instances that experience significant errors when generating the AWR snapshots. In such cases, ADDM will analyze the largest subset of instances that did not experience these problems.

The primary interface for diagnostic monitoring is Oracle Enterprise Manager. Whenever possible, you should run ADDM using Oracle Enterprise Manager, as described in Oracle Database 2 Day + Performance Tuning Guide. If Oracle Enterprise Manager is unavailable, you can run ADDM using the DBMS_ADDM package. In order to run the DBMS_ADDM APIs, the user must be granted the ADVISOR privilege.

Running ADDM in Database Mode

For Oracle RAC configurations, you can run ADDM in Database mode to analyze all instances of the databases. For single-instance configurations, you can still run ADDM in Database mode; ADDM will simply behave as if running in Instance mode.

To run ADDM in Database mode, use the DBMS_ADDM.ANALYZE_DB procedure:

```
BEGIN
DBMS_ADDM.ANALYZE_DB (
  task_name          IN OUT VARCHAR2,
  begin_snapshot     IN     NUMBER,
  end_snapshot       IN     NUMBER,
  db_id              IN     NUMBER := NULL);
END;
/
```

The task_name parameter specifies the name of the analysis task that will be created. The begin_snapshot parameter specifies the snapshot number of the beginning snapshot in the analysis period. The end_snapshot parameter specifies the snapshot number of the ending snapshot in the analysis period. The db_id parameter specifies the database identifier of the database that will be analyzed. If unspecified, this parameter defaults to the database identifier of the database to which you are currently connected.

The following example creates an ADDM task in database analysis mode, and executes it to diagnose the performance of the entire database during the time period defined by snapshots 137 and 145:

```
VAR tname VARCHAR2(30);
BEGIN
  :tname := 'ADDM for 7PM to 9PM';
  DBMS_ADDM.ANALYZE_DB(:tname, 137, 145);
END;
/
```

Running ADDM in Instance Mode

To analyze a particular instance of the database, you can run ADDM in Instance mode. To run ADDM in Instance mode, use the DBMS_ADDM.ANALYZE_INST procedure:

```
BEGIN
DBMS_ADDM.ANALYZE_INST (
   task_name           IN OUT VARCHAR2,
   begin_snapshot      IN     NUMBER,
   end_snapshot        IN     NUMBER,
   instance_number     IN     NUMBER := NULL,
   db_id               IN     NUMBER := NULL);
END;
/
```

The task_name parameter specifies the name of the analysis task that will be created. The begin_snapshot parameter specifies the snapshot number of the beginning snapshot in the analysis period. The end_snapshot parameter specifies the snapshot number of the ending snapshot in the analysis period. The instance_number parameter specifies the instance number of the instance that will be analyzed. If unspecified, this parameter defaults to the instance number of the instance to which you are currently connected. The db_id parameter specifies the database identifier of the database that will be analyzed. If unspecified, this parameter defaults to the database identifier of the database to which you are currently connected.

The following example creates an ADDM task in instance analysis mode, and executes it to diagnose the performance of instance number 1 during the time period defined by snapshots 137 and 145:

```
VAR tname VARCHAR2(30);
BEGIN
  :tname := 'my ADDM for 7PM to 9PM';
  DBMS_ADDM.ANALYZE_INST(:tname, 137, 145, 1);
END;
/
```

Running ADDM in Partial Mode

To analyze a subset of all database instances, you can run ADDM in Partial mode. To run ADDM in Partial mode, use the DBMS_ADDM.ANALYZE_PARTIAL procedure:

```
BEGIN
```

```
DBMS_ADDM.ANALYZE_PARTIAL (
   task_name          IN OUT VARCHAR2,
   instance_numbers   IN     VARCHAR2,
   begin_snapshot     IN     NUMBER,
   end_snapshot       IN     NUMBER,
   db_id              IN     NUMBER := NULL);
END;
/
```

The task_name parameter specifies the name of the analysis task that will be created. The instance_numbers parameter specifies a comma-delimited list of instance numbers of instances that will be analyzed. The begin_snapshot parameter specifies the snapshot number of the beginning snapshot in the analysis period. The end_snapshot parameter specifies the snapshot number of the ending snapshot in the analysis period. The db_id parameter specifies the database identifier of the database that will be analyzed. If unspecified, this parameter defaults to the database identifier of the database to which you are currently connected.

The following example creates an ADDM task in partial analysis mode, and executes it to diagnose the performance of instance numbers 1, 2, and 4, during the time period defined by snapshots 137 and 145:

```
VAR tname VARCHAR2(30);
BEGIN
  :tname := 'my ADDM for 7PM to 9PM';
  DBMS_ADDM.ANALYZE_PARTIAL(:tname, '1,2,4', 137, 145);
END;
/
```

Displaying an ADDM Report

To display a text report of an executed ADDM task, use the DBMS_ADDM.GET_REPORT function:

```
DBMS_ADDM.GET_REPORT (
   task_name          IN VARCHAR2
  RETURN CLOB);
```

The following example displays a text report of the ADDM task specified by its task name using the tname variable:

```
SET LONG 1000000 PAGESIZE 0;
SELECT DBMS_ADDM.GET_REPORT(:tname) FROM DUAL;
```

Note that the return type of a report is a CLOB, formatted to fit line size of 80.

Views with ADDM Information

Typically, you should view output and information from ADDM using Oracle Enterprise Manager or ADDM reports.

However, you can display ADDM information through the DBA_ADVISOR views. This group of views includes:

- DBA_ADVISOR_FINDINGS

  This view displays all the findings discovered by all advisors. Each finding is displayed with an associated finding ID, name, and type. For tasks with multiple executions, the name of each task execution associated with each finding is also listed.

- DBA_ADDM_FINDINGS

  This view contains a subset of the findings displayed in the related DBA_ADVISOR_FINDINGS view. This view only displays the ADDM findings discovered by all advisors. Each ADDM finding is displayed with an associated finding ID, name, and type.

- DBA_ADVISOR_FINDING_NAMES

  List of all finding names registered with the advisor framework.

- DBA_ADVISOR_RECOMMENDATIONS

  This view displays the results of completed diagnostic tasks with recommendations for the problems identified in each execution. The recommendations should be reviewed in the order of the RANK column, as this relays the magnitude of the problem for the recommendation. The BENEFIT column displays the benefit to the system you can expect after the recommendation is performed. For tasks with multiple executions, the name of each task execution associated with each advisor task is also listed.

- DBA_ADVISOR_TASKS

  This view provides basic information about existing tasks, such as the task ID, task name, and when the task was created. For tasks with multiple executions, the name and type of the last or current execution associated with each advisor task is also listed.

SQL Tuning Advisor

The SQL Tuning Advisor takes one or more SQL statements as an input and invokes the Automatic Tuning Optimizer to perform SQL tuning on the statements. The output of the SQL Tuning Advisor is in the form of an advice or recommendations, along with a rationale for each recommendation and its expected benefit. The recommendation relates to collection of statistics on objects, creation of new indexes, restructuring of the SQL statement, or creation of a SQL profile. You can choose to accept the recommendation to complete the tuning of the SQL statements.

Oracle Database can automatically tune SQL statements by identifying problematic SQL statements and implementing tuning recommendations using the SQL Tuning Advisor during system maintenance windows. You can also run the SQL Tuning Advisor selectively on a single or a set of SQL statements that have been identified as problematic.

Automatic SQL Tuning Advisor

Oracle Database automatically runs the SQL Tuning Advisor on selected high-load SQL statements from the Automatic Workload Repository (AWR) that qualify as tuning candidates. This task, called Automatic SQL Tuning, runs in the default maintenance windows on a nightly basis. You can customize attributes of the maintenance windows, including start and end time, frequency, and days of the week.

Once automatic SQL tuning begins, which by default runs for at most one hour during a maintenance window, the following steps are performed:

1. Identify SQL candidates in the AWR for tuning.

   Oracle Database analyzes statistics in the AWR and generates a list of potential SQL statements that are eligible for tuning. These statements include repeating high-load statements that have a significant impact on the system. Only SQL statements that have an execution plan with a high potential for improvement will be tuned. Recursive SQL and statements that have been tuned recently (in the last month) are ignored, as are parallel queries, DMLs, DDLs, and SQL statements with performance problems that are caused by concurrency issues. The SQL statements that are selected as candidates are then ordered based on their performance impact. The performance impact of a SQL statement is calculated by summing the CPU time and the I/O times captured in the AWR for that SQL statement in the past week.

2. Tune each SQL statement individually by calling the SQL Tuning Advisor.

   During the tuning process, all recommendation types are considered and reported, but only SQL profiles can be implemented automatically.

3. Test SQL profiles by executing the SQL statement.

   If a SQL profile is recommended, test the new SQL profile by executing the SQL statement both with and without the SQL profile. If the performance improvement improves at least threefold, the SQL profile will be accepted (when the ACCEPT_SQL_PROFILES task parameter is set to TRUE). Otherwise, only the recommendation to create a SQL profile will be reported in the automatic SQL tuning reports.

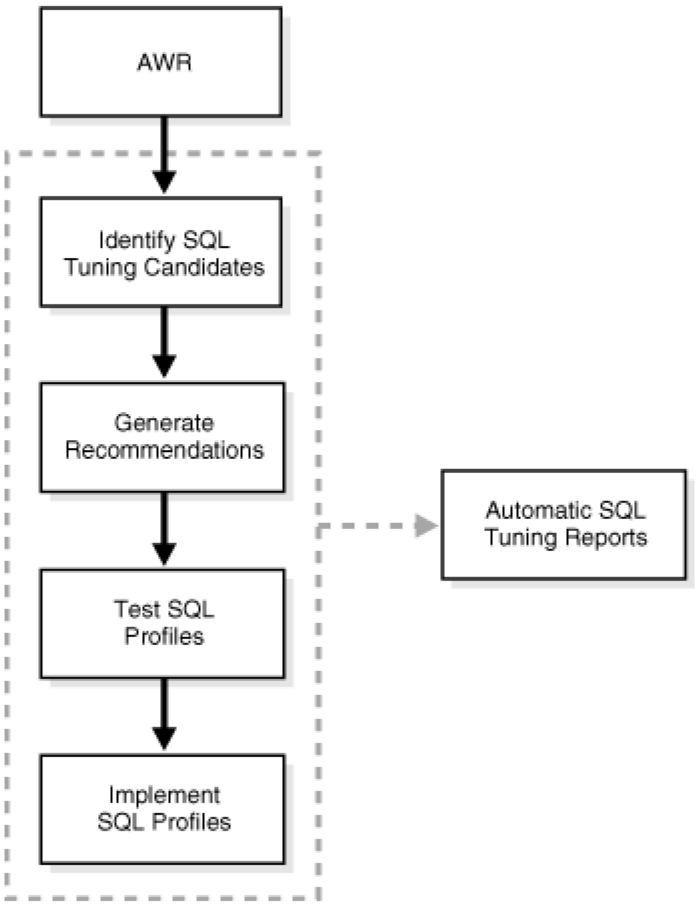
4. Optionally implement the SQL profiles provided they meet the criteria of threefold performance improvement.

> Note that other factors are considered when deciding whether or not to implement the SQL profile. For example, a SQL profile is not implemented if the objects referenced in the SQL statement have stale optimizer statistics. You can identify which SQL profiles have been implemented automatically as their type will be set to AUTO in the DBA_SQL_PROFILES view.
>
> If SQL plan management is used and there is already an existing plan baseline for the SQL statement, a new plan baseline will be added when a SQL profile is created. As a result, the new and improved SQL execution plan will be used by the optimizer immediately after the SQL profile is created.

At any time during or after the automatic SQL tuning process, you can view the results using the automatic SQL tuning report. This report describes in detail all the SQL statements that were analyzed, the recommendations generated, and the SQL profiles that were automatically implemented.

Figure 1 illustrates the steps performed by Oracle Database during the automatic SQL tuning process.

Figure 1 Automatic SQL Tuning



Enabling and Disabling Automatic SQL Tuning

Automatic SQL tuning runs as part of the automated maintenance tasks infrastructure.

To enable automatic SQL tuning, use the ENABLE procedure in the DBMS_AUTO_TASK_ADMIN package:

```
BEGIN
  DBMS_AUTO_TASK_ADMIN.ENABLE(
    client_name => 'sql tuning advisor',
    operation => NULL,
    window_name => NULL);
END;
/
```

To disable automatic SQL tuning, use the DISABLE procedure in the DBMS_AUTO_TASK_ADMIN package:

```
BEGIN
  DBMS_AUTO_TASK_ADMIN.DISABLE(
    client_name => 'sql tuning advisor',
    operation => NULL,
    window_name => NULL);
END;
/
```

You can pass a specific window name using the window_name parameter to enable or disable the task in certain maintenance windows only.

Setting the STATISTICS_LEVEL parameter to BASIC will disable automatic statistics gathering by the AWR and, as a result, also disable automatic SQL tuning.

Configuring Automatic SQL Tuning

The behavior of the automatic SQL tuning task can be configured using the DBMS_SQLTUNE package. To use the APIs, the user needs at least the ADVISOR privilege.

In addition to configuring the standard behavior of the SQL Tuning Advisor, the DBMS_SQLTUNE package enables you to configure automatic SQL tuning by specifying the task parameters using the SET_TUNING_TASK_PARAMETER procedure. Because the automatic tuning task is owned by SYS, only the SYS user can set the task parameters.

Table 1: SET_TUNING_TASK_PARAMETER Automatic SQL Tuning Parameters

Parameter	**Description**
ACCEPT_SQL_PROFILE	Specifies whether to accept SQL profiles automatically.
MAX_SQL_PROFILES_PER_EXEC	Specifies the limit of SQL profiles that are accepted for each

Parameter	**Description**
	automatic SQL tuning task. Consider setting the limit of SQL profiles that are accepted for each automatic SQL tuning task based on the acceptable level of changes that can be made to the system on a daily basis.
MAX_AUTO_SQL_PROFILES	Specifies the limit of SQL profiles that are accepted in total.
EXECUTION_DAYS_TO_EXPIRE	Specifies the number of days for which to save the task history in the advisor framework schema. By default, the task history is saved for 30 days before it expires.

To configure automatic SQL tuning, run the SET_TUNING_TASK_PARAMETER procedure in the DBMS_SQLTUNE package:

```
BEGIN
  DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER(
    task_name => 'SYS_AUTO_SQL_TUNING_TASK',
    parameter => 'ACCEPT_SQL_PROFILES', value => 'TRUE');
END;
/
```

In this example, the automatic SQL tuning task is configured to automatically accept SQL profiles recommended by the SQL Tuning Advisor.

Viewing Automatic SQL Tuning Reports

The automatic SQL tuning report is generated using the DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK function and contains information about all executions of the automatic SQL tuning task. To run this report, you need the ADVISOR privilege and SELECT privileges on the DBA_ADVISOR views. Unlike the standard SQL tuning report generated using the DBMS_SQLTUNE.REPORT_TUNING_TASK function, which only contains information about a single task execution of the SQL Tuning Advisor, the automatic SQL tuning report contains information about multiple executions of the automatic SQL tuning task.

To view the automatic SQL tuning report, run the REPORT_AUTO_TUNING_TASK function in the DBMS_SQLTUNE package:

```
variable my_rept CLOB;
BEGIN
  :my_rept :=DBMS_SQLTUNE.REPORT_AUTO_TUNING_TASK(
    begin_exec => NULL,
    end_exec => NULL,
    type => 'TEXT',
    level => 'TYPICAL',
    section => 'ALL',
    object_id => NULL,
    result_limit => NULL);
```

```
END;
/

print :my_rept
```

In this example, a text report is generated to display all SQL statements that were analyzed in the most recent execution, including recommendations that were not implemented, and all sections of the report are included.

Depending on the sections that were included in the report, you can view information about the automatic SQL tuning task in the following sections of the report:

- General information

  The general information section provides a high-level description of the automatic SQL tuning task, including information about the inputs given for the report, the number of SQL statements tuned during the maintenance, and the number of SQL profiles that were created

- Summary

  The summary section lists the SQL statements (by their SQL identifiers) that were tuned during the maintenance window and the estimated benefit of each SQL profile, or their actual execution statistics after test executing the SQL statement with the SQL profile

- Tuning findings

  This section contains the following information about each SQL statement analyzed by the SQL Tuning Advisor:

  - All findings associated with each SQL statement
  - Whether the profile was accepted on the system, and why
  - Whether the SQL profile is currently enabled on the system
  - Detailed execution statistics captured when testing the SQL profile
- Explain plans

  This section shows the old and new explain plans used by each SQL statement analyzed by the SQL Tuning Advisor.

- Errors

  This section lists all errors encountered by the automatic SQL tuning task.

Tuning Options

The SQL Tuning Advisor provides options to manage the scope and duration of a tuning task. The scope of a tuning task can be set to limited or comprehensive.

www.csitportal.com

- If the limited option is chosen, the SQL Tuning Advisor produces recommendations based on statistics checks, access path analysis, and SQL structure analysis. SQL Profile recommendations are not generated.
- If the comprehensive option is selected, the SQL Tuning Advisor carries out all the analysis it performs under limited scope plus SQL Profiling. With the comprehensive option you can also specify a time limit for the tuning task, which by default is 30 minutes.

Advisor Output

After analyzing the SQL statements, the SQL Tuning Advisor provides advice on optimizing the execution plan, the rationale for the proposed optimization, the estimated performance benefit, and the command to implement the advice. You simply have to choose whether or not to accept the recommendations to optimize the SQL statements.

Running the SQL Tuning Advisor

The recommended interface for running the SQL Tuning Advisor is the Oracle Enterprise Manager. If Oracle Enterprise Manager is unavailable, you can run the SQL Tuning Advisor using procedures in the DBMS_SQLTUNE package. To use the APIs, the user must be granted specific privileges.

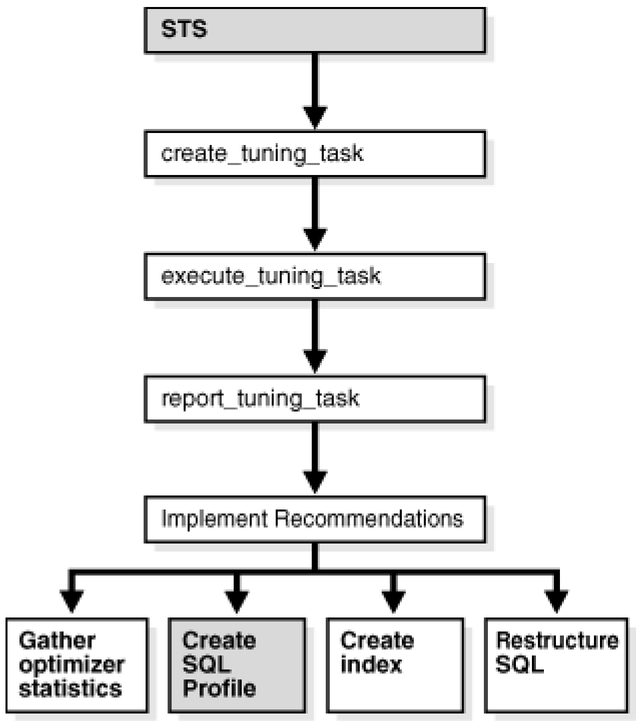
Running SQL Tuning Advisor using DBMS_SQLTUNE package is a multi-step process:

1. Create a SQL Tuning Set (if tuning multiple SQL statements)
2. Create a SQL tuning task
3. Execute a SQL tuning task
4. Display the results of a SQL tuning task
5. Implement recommendations as appropriate

A SQL tuning task can be created for a single SQL statement. For tuning multiple statements, a SQL Tuning Set (STS) has to be first created. An STS is a database object that stores SQL statements along with their execution context. An STS can be created manually using command line APIs or automatically using Oracle Enterprise Manager. Figure 2: shows the steps involved when running the SQL Tuning Advisor using the DBMS_SQLTUNE package.

Figure 2 SQL Tuning Advisor APIs

www.csitportal.com



Creating a SQL Tuning Task

You can create tuning tasks from the text of a single SQL statement, a SQL Tuning Set containing multiple statements, a SQL statement selected by SQL identifier from the cursor cache, or a SQL statement selected by SQL identifier from the Automatic Workload Repository.

For example, to use the SQL Tuning Advisor to optimize a specified SQL statement text, you need to create a tuning task with the SQL statement passed as a CLOB argument. For the following PL/SQL code, the user HR has been granted the ADVISOR privilege and the function is run as user HR on the employees table in the HR schema.

```
DECLARE
 my_task_name VARCHAR2(30);
 my_sqltext   CLOB;
BEGIN
 my_sqltext := 'SELECT /*+ ORDERED */ * '                    ||
             'FROM employees e, locations l, departments d ' ||
             'WHERE e.department_id = d.department_id AND '  ||
                  'l.location_id = d.location_id AND '      ||
                  'e.employee_id < :bnd';

 my_task_name := DBMS_SQLTUNE.CREATE_TUNING_TASK(
        sql_text    => my_sqltext,
        bind_list   => sql_binds(anydata.ConvertNumber(100)),
        user_name   => 'HR',
        scope       => 'COMPREHENSIVE',
        time_limit  => 60,
        task_name   => 'my_sql_tuning_task',
    description => 'Task to tune a query on a specified employee');
END;
/
```

In this example, 100 is the value for bind variable :bnd passed as function argument of type SQL_BINDS, HR is the user under which the CREATE_TUNING_TASK function analyzes the SQL statement, the scope is set to COMPREHENSIVE which means that the advisor also performs SQL Profiling analysis, and 60 is the maximum time in seconds that the function can run. In addition, values for task name and description are provided.

The CREATE_TUNING_TASK function returns the task name that you have provided or generates a unique task name. You can use the task name to specify this task when using other APIs. To view the task names associated with a specific owner, you can run the following:

```
SELECT task_name FROM DBA_ADVISOR_LOG WHERE owner = 'HR';
```

Configuring a SQL Tuning Task

You can fine tune a SQL tuning task after it has been created by configuring its parameters using the SET_TUNING_TASK_PARAMETER procedure in the DBMS_SQLTUNE package:

```
BEGIN
  DBMS_SQLTUNE.SET_TUNING_TASK_PARAMETER(
    task_name => 'my_sql_tuning_task',
    parameter => 'TIME_LIMIT', value => 300);
END;
/
```

In this example, the maximum time that the SQL tuning task can run is changed to 300 seconds.

Table 2 lists the parameters that can be configured using the SET_TUNING_TASK_PARAMETER procedure.

Table 2 SET_TUNING_TASK_PARAMETER Procedure Parameters

Parameter	**Description**
MODE	Specifies the scope of the tuning task: • LIMITED takes approximately 1 second to tune each SQL statement but does not recommend a SQL profile • COMPREHENSIVE performs a complete analysis and recommends a SQL profile, when appropriate, but may take much longer.
USERNAME	Username under which the SQL statement will be parsed
DAYS_TO_EXPIRE	Number of days before the task is deleted
DEFAULT_EXECUTION_TYPE	Default execution type if not specified by the

Parameter	**Description**
	EXECUTE_TUNING_TASK function when the task is executed
TIME_LIMIT	Time limit (in number of seconds) before the task times out
LOCAL_TIME_LIMIT	Time limit (in number of seconds) for each SQL statement
TEST_EXECUTE	Determines if the SQL Tuning Advisor will test execute the SQL statements to verify the recommendation benefit: • FULL - Test executes SQL statements for as much of the local time limit as necessary • AUTO - Test executes SQL statements using an automatic time limit • OFF - Will not test execute SQL statements
BASIC_FILTER	Basic filter used for SQL tuning set
OBJECT_FILTER	Object filter used for SQL tuning set
PLAN_FILTER	Plan filter used for SQL tuning set
RANK_MEASURE1	First ranking measure used for SQL tuning set
RANK_MEASURE2	Second ranking measure used for SQL tuning set
RANK_MEASURE3	Third ranking measure used for SQL tuning set
RESUME_FILTER	Extra filter used for SQL tuning set (besides BASIC_FILTER)
SQL_LIMIT	Maximum number of SQL statements to tune
SQL_PERCENTAGE	Percentage filter of statements from SQL tuning set

Executing a SQL Tuning Task

After you have created a tuning task, you need to execute the task and start the tuning process. For example:

```
BEGIN
  DBMS_SQLTUNE.EXECUTE_TUNING_TASK( task_name => 'my_sql_tuning_task' );
END;
/
```

Like any other SQL Tuning Advisor task, you can also execute the automatic tuning task SYS_AUTO_SQL_TUNING_TASK using the EXECUTE_TUNING_TASK API. The SQL Tuning Advisor will perform the same analysis and actions as it would when run automatically. You can also pass an execution name to the API to name the new execution.

Checking the Status of a SQL Tuning Task

You can check the status of the task by reviewing the information in the USER_ADVISOR_TASKS view or check execution progress of the task in the V$SESSION_LONGOPS view. For example:

```
SELECT status FROM USER_ADVISOR_TASKS WHERE task_name = 'my_sql_tuning_task';
```

Checking the Progress of the SQL Tuning Advisor

You can check the execution progress of the SQL Tuning Advisor in the V$ADVISOR_PROGRESS view. For example:

```
SELECT sofar, totalwork FROM V$ADVISOR_PROGRESS WHERE user_name = 'HR' AND task_name =
'my_sql_tuning_task';
```

Displaying the Results of a SQL Tuning Task

After a task has been executed, you display a report of the results with the REPORT_TUNING_TASK function. For example:

```
SET LONG 1000
SET LONGCHUNKSIZE 1000
SET LINESIZE 100
SELECT DBMS_SQLTUNE.REPORT_TUNING_TASK( 'my_sql_tuning_task')
  FROM DUAL;
```

The report contains all the findings and recommendations of the SQL Tuning Advisor. For each proposed recommendation, the rationale and benefit is provided along with the SQL commands needed to implement the recommendation.

Additional Operations on a SQL Tuning Task

You can use the following APIs for managing SQL tuning tasks:

- INTERRUPT_TUNING_TASK to interrupt a task while executing, causing a normal exit with intermediate results
- RESUME_TUNING_TASK to resume a previously interrupted task
- CANCEL_TUNING_TASK to cancel a task while executing, removing all results from the task
- RESET_TUNING_TASK to reset a task while executing, removing all results from the task and returning the task to its initial state
- DROP_TUNING_TASK to drop a task, removing all results associated with the task

SQL Tuning Sets

A SQL Tuning Set (STS) is a database object that includes one or more SQL statements along with their execution statistics and execution context, and could include a user priority ranking. The SQL statements can be loaded into a SQL Tuning Set from different SQL sources, such as the Automatic Workload Repository, the cursor cache, or custom SQL provided by the user. An STS includes:

- A set of SQL statements
- Associated execution context, such as user schema, application module name and action, list of bind values, and the cursor compilation environment
- Associated basic execution statistics, such as elapsed time, CPU time, buffer gets, disk reads, rows processed, cursor fetches, the number of executions, the number of complete executions, optimizer cost, and the command type
- Associated execution plans and row source statistics for each SQL statement (optional)

SQL statements can be filtered using the application module name and action, or any of the execution statistics. In addition, the SQL statements can be ranked based on any combination of execution statistics.

A SQL Tuning Set can be used as input to the SQL Tuning Advisor, which performs automatic tuning of the SQL statements based on other input parameters specified by the user. SQL Tuning Sets are transportable across databases and can be exported from one system to another, allowing for the transfer of SQL workloads between databases for remote performance diagnostics and tuning. When poorly performing SQL statements are encountered on a production system, it may not be desirable for developers to perform their investigation and tuning activities on the production system directly. This feature allows the DBA to transport the problematic SQL statements to a test system where the developers can safely analyze and tune them. To transport SQL Tuning Sets, use the DBMS_SQLTUNE package procedures.

Creating a SQL Tuning Set

The CREATE_SQLSET procedure is used to create an empty STS object in the database. For example, the following procedure creates an STS object that could be used to tune I/O intensive SQL statements during a specific period of time:

```
BEGIN
  DBMS_SQLTUNE.CREATE_SQLSET(
    sqlset_name => 'my_sql_tuning_set',
    description  => 'I/O intensive workload');
END;
/
```

where my_sql_tuning_set is the name of the STS in the database and 'I/O intensive workload' is the description assigned to the STS.

Loading a SQL Tuning Set

The LOAD_SQLSET procedure populates the STS with selected SQL statements. The standard sources for populating an STS are the workload repository, another STS, or the cursor cache. For both the workload repository and STS, predefined table functions can be used to select columns from the source to populate a new STS.

In the following example, procedure calls are used to load my_sql_tuning_set from an AWR baseline called peak baseline. The data has been filtered to select only the top 30 SQL statements ordered by elapsed time. First a ref cursor is opened to select from the specified baseline. Next the statements and their statistics are loaded from the baseline into the STS.

```
DECLARE
 baseline_cursor DBMS_SQLTUNE.SQLSET_CURSOR;
BEGIN
 OPEN baseline_cursor FOR
    SELECT VALUE(p)
    FROM TABLE (DBMS_SQLTUNE.SELECT_WORKLOAD_REPOSITORY(
                'peak baseline',
                 NULL, NULL,
                 'elapsed_time',
                 NULL, NULL, NULL,
                 30)) p;

    DBMS_SQLTUNE.LOAD_SQLSET(
            sqlset_name     => 'my_sql_tuning_set',
            populate_cursor => baseline_cursor);
END;
/
```

Displaying the Contents of a SQL Tuning Set

The SELECT_SQLSET table function reads the contents of the STS. After an STS has been created and populated, you can browse the SQL in the STS using different filtering criteria. The SELECT_SQLSET procedure is provided for this purpose.

In the following example, the SQL statements in the STS are displayed for statements with a disk-reads to buffer-gets ratio greater than or equal to 75%.

```
SELECT * FROM TABLE(DBMS_SQLTUNE.SELECT_SQLSET(
   'my_sql_tuning_set',
   '(disk_reads/buffer_gets) >= 0.75'));
```

Additional details of the SQL Tuning Sets that have been created and loaded can also be displayed with DBA views, such as DBA_SQLSET, DBA_SQLSET_STATEMENTS, and DBA_SQLSET_BINDS.

Modifying a SQL Tuning Set

SQL statements can be updated and deleted from a SQL Tuning Set based on a search condition. In the following example, the DELETE_SQLSET procedure deletes SQL statements from my_sql_tuning_set that have been executed less than fifty times.

```
BEGIN
  DBMS_SQLTUNE.DELETE_SQLSET(
      sqlset_name  => 'my_sql_tuning_set',
      basic_filter => 'executions < 50');
END;
```

/

Transporting a SQL Tuning Set

SQL Tuning Sets can be transported to another system by first exporting the STS from one system to a staging table, then importing the STS from the staging table into another system.

To transport a SQL Tuning Set:

1. Use the CREATE_STGTAB_SQLSET procedure to create a staging table where the SQL Tuning Sets will be exported.

   The following example shows how to create a staging table named staging_table. Table names are case-sensitive.

   ```
   BEGIN
     DBMS_SQLTUNE.CREATE_STGTAB_SQLSET( table_name => 'staging_table' );
   END;
   /
   ```

2. Use the PACK_STGTAB_SQLSET procedure to export SQL Tuning Sets into the staging table.

   The following example shows how to export a SQL Tuning Set named my_sts to the staging table.

   ```
   BEGIN
     DBMS_SQLTUNE.PACK_STGTAB_SQLSET(
         sqlset_name  => 'my_sts',
         staging_table_name => 'staging_table');
   END;
   /
   ```

3. Move the staging table to the system where the SQL Tuning Sets will be imported using the mechanism of choice (such as datapump or database link).
4. On the system where the SQL Tuning Sets will be imported, use the UNPACK_STGTAB_SQLSET procedure to import SQL Tuning Sets from the staging table.

   The following example shows how to import SQL Tuning Sets contained in the staging table.

   ```
   BEGIN
     DBMS_SQLTUNE.UNPACK_STGTAB_SQLSET(
         sqlset_name  => '%',
         replace  => TRUE,
         staging_table_name => 'staging_table');
   END;
   /
   ```

www.csitportal.com

Dropping a SQL Tuning Set

The DROP_SQLSET procedure is used to drop an STS that is no longer needed. For example:

```
BEGIN
  DBMS_SQLTUNE.DROP_SQLSET( sqlset_name => 'my_sql_tuning_set' );
END;
/
```

Additional Operations on SQL Tuning Sets

You can use the following APIs to manage an STS:

- Updating the attributes of SQL statements in an STS

  The UPDATE_SQLSET procedure updates the attributes of SQL statements (such as PRIORITY or OTHER) in an existing STS identified by STS name and SQL identifier.

- Capturing the full system workload

  The CAPTURE_CURSOR_CACHE_SQLSET function enables the capture of the full system workload by repeatedly polling the cursor cache over a specified interval. This function is a lot more efficient than repeatedly using the SELECT_CURSOR_CACHE and LOAD_SQLSET procedures to capture the cursor cache over an extended period of time. This function effectively captures the entire workload, as opposed to the AWR—which only captures the workload of high-load SQL statements—or the LOAD_SQLSET procedure, which accesses the data source only once.

- Adding and removing a reference to an STS

  The ADD_SQLSET_REFERENCE function adds a new reference to an existing STS to indicate its use by a client. The function returns the identifier of the added reference. The REMOVE_SQLSET_REFERENCE procedure is used to deactivate an STS to indicate it is no longer used by the client.

SQL Tuning Information Views

This section summarizes the views that you can display to review information that has been gathered for tuning the SQL statements. You need DBA privileges to access these views.

- Advisor information views, such as DBA_ADVISOR_TASKS, DBA_ADVISOR_EXECUTIONS, DBA_ADVISOR_FINDINGS, DBA_ADVISOR_RECOMMENDATIONS, and DBA_ADVISOR_RATIONALE views.
- SQL tuning information views, such as DBA_SQLTUNE_STATISTICS, DBA_SQLTUNE_BINDS, and DBA_SQLTUNE_PLANS views.
- SQL Tuning Set views, such as DBA_SQLSET, DBA_SQLSET_BINDS, DBA_SQLSET_STATEMENTS, and DBA_SQLSET_REFERENCES views.

- Information on captured execution plans for statements in SQL Tuning Sets are displayed in the DBA_SQLSET_PLANS and USER_SQLSET_PLANS views.
- SQL Profile information is displayed in the DBA_SQL_PROFILES view.

  The TYPE parameter shows if the SQL profile was created manually by the SQL Tuning Advisor (if TYPE = MANUAL) or automatically by automatic SQL tuning (if TYPE = AUTO).

- Advisor execution progress information is displayed in the V$ADVISOR_PROGRESS view.
- Dynamic views containing information relevant to the SQL tuning, such as VSQL, VSQLAREA, V$SQLSTATS, and V$SQL_BINDS views.

Introduction to Virtual Private Databases

Oracle Virtual Private Database (VPD) enables you to create security policies to control database access at the row and column level. Essentially, Oracle Virtual Private Database adds a dynamic WHERE clause to a SQL statement that is issued against the table, view, or synonym to which an Oracle Virtual Private Database security policy was applied.

Oracle Virtual Private Database enforces security, to a fine level of granularity, directly on database tables, views, or synonyms. Because you attach security policies directly to these database objects, and the policies are automatically applied whenever a user accesses data, there is no way to bypass security.

You can apply Oracle Virtual Private Database policies to SELECT, INSERT, UPDATE, INDEX, and DELETE statements.

To implement Oracle Virtual Private Database, you must create a function to generate the dynamic WHERE clause, and a policy to attach this function to the objects that you want to protect.
I will explain that definition using example below.

Sample Case

- We have department (10,20,30) in EMP table and want
  User "user10" to select employees in department 10 only.
  User "user20" to select employees in department 20only.
  User "user30" to select employees in department 30 only.

Step 1 : Create Policy Group

```
BEGIN
  SYS.DBMS_RLS.CREATE_POLICY_GROUP
  (
   object_schema => 'SCOTT'
   ,object_name => 'EMP'
   ,policy_group => 'EMP_POLICIES'
  );
END;
```

Step 2 : Create Policy Function

Policy function will return varchar2 and that is predicate that is added to where clause at table.
Function signature must be as done at function example.
This can be in a package or a stand-alone function.

```
CREATE OR REPLACE FUNCTION auth_emps (
   schema_var   IN   VARCHAR2,                                   --required
   table_var    IN   VARCHAR2                                  --required
)
   RETURN VARCHAR2
IS
   return_val   VARCHAR2 (400);
BEGIN
   return_val :=
      CASE USER
         WHEN 'USER10'
            THEN 'DEPTNO = 10'
         WHEN 'USER20'
            THEN 'DEPTNO = 20'
         WHEN 'USER30'
            THEN 'DEPTNO = 30'
         ELSE NULL
      END;
   RETURN return_val;
END auth_emps;
```

Step 3 : Create the policy

```
BEGIN
  SYS.DBMS_RLS.ADD_GROUPED_POLICY
    (
      Object_schema         => 'SCOTT'
     ,Object_name           => 'EMP'
     ,policy_group          => 'EMP_POLICIES'
     ,policy_name           => 'SCOTT_EMPS'
     ,function_schema       => 'SCOTT'
     ,policy_function       => 'AUTH_EMPS'
     ,statement_types       => 'SELECT '
     ,policy_type           => dbms_rls.dynamic
     ,long_predicate        => FALSE
     ,sec_relevant_cols     => 'EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO'
     ,sec_relevant_cols_opt => NULL
     ,update_check          => FALSE
     ,enable                => TRUE
    );
```

END;

Step 4 : Test
Let's now connect using user10 the result set will be only employees that in Department 10.

Recovering Database from noncritical losses

Recovering from Loss of a Temporary Tablespace

It is impossible to backup or restore temporary tablespaces. If damaged, they can be replaced, instead of restored.

Loss of undo data is a critical data loss.

Temporary data in temporary tablespaces is data that exists for one database session or less. The data is also private to the session that created it.

Temporary data comes in two forms:

Sort data is generated when an operation requiring rows to be sorted occurs and it can't fit in memory. Examples are ORDER BY or creating indices.

Global temporary tables are used globally however, each session can only see the rows that it inserted. It also depends on operations not fitting into memory (ex. create global temporary table gt1 as select * from hr.employees;).

Working with temporary data never produces undo or redo data.

If a tempfile is not available at startup, the database will still open. The problem won't become apparent until temporary space is needed or you check the alert log. (ex. create global temporary table gt1 …)

Damage to a Tempfile

Temporary tablespaces are very different from the tablespaces / datafiles that make up permanent tablespaces. Temporary tablespaces are not needed to OPEN the database, while all permanent tablespaces are. Missing temporary tablespaces will produce messages in the alert log and throw errors to users who require temp space.

Temporary tablespaces are also not written to by the DBWn processes, they are written to by the server processes servicing the sessions that need temp space.

Restoring a Temporary Tablespace

SQL> alter tablespace temp_tbs1 add tempfile '/opt/…/temp_tbs1.dbf' size 100m;

add new datafile to existing damaged temp tablespace

SQL> alter database tempfile '/…/old_tmp_tbs.dbf' offline;

take damaged datafile offline

SQL> alter database tempfile '/…/old_tmp_tbs.dbf' drop;

drop damaged datafile

Recovering from Loss of an Online Redo Log File

An Oracle db requires at least two online file groups, each with a valid member.

The log with the highest sequence number (according to v$log) is the CURRENT group.

An ACTIVE group is not current but still has data referrring to blocks in the db buffer cache (dirty) not yet written to the datafiles by the DBWR. If the instance failed, both the CURRENT and ACTIVE groups would be needed for recovery since they both contain references to dirty blocks in the db buffer cache. An INACTIVE group contains no dirty references.

Damage to a multiplexed redo log member won't affect the instance's OPEN status.

Damage to all copies of the current logfile group will cause the instance to terminate immediately. It will also terminate at log switch if all copies of the log group being made current are damaged.

To recreate a damaged log file member there are two choices:

Drop the damaged member and add a replacement

alter database drop logfile member '/u01/…';

delete the file from the operating system

alter database add logfile member '/u01/…' to group 1;

Clear the log group

alter database clear logfile group 2;

Recovering from Loss of an Index Tablespace

If an index is not available, the table will be locked for nearly all DML operations: (no inserts, deletes or updates).

Index data can be regenerated from the base tables and is therefore considered noncritical.

www.csitportal.com

It is generally considered good practice to create a tablespace specifically for your indexes. If the index tablespace and table tablespace are on different disks, there will be less contention on the data, as well.

Recreating an index tablespace:

Take damaged tablespace offline

Determine which indexes were in the damaged tablespace

select owner, segment_name from dba_segments where tablespace_name='INDX' and segment_type='INDEX';

Drop the tablespace and delete the files from the OS.

drop tablespace INDX including contents and datafiles;

Create a new tablespace.

Generate all the indexes in it.

Regenerating indexes:

Many applications have GUI options to rebuild indexes.

There are also rebuild index scripts often contained in the documentation.

If there is no documentation, you can query the Data Dictionary (join DBA_INDEXES and DBA_IND_COLUMNS) to find which columns in which tables were indexed, which type of index (B*Tree or bitmap) or other characteristics like UNIQUE or COMPRESSED.

The Data Dictionary will delete index information with the "including contents" clause of the"drop tablespace" statement.

Another option is to use datapump to extract all index definitions and reimport them.

IOTs are indexes (not tables) yet they must be backed up like regular heap tables. Backing up an index tablespace is optional, however. To decide you should do a cost benefit analysis of how long it would take to restore vs. how long it would take to recreate the indexes.

If an index is not available due to media damage, any DML against the indexed columns will fail. Queries against the indexed columns will also fail. Even DML against constrained columns in a child table (if referencing a parent table with a missing index) will fail.

Recovering from Loss of a Read-Only Tablespace

When the command ALTER TABLESPACE .. READ ONLY is issued, the datafiles are checkpointed (db buffer cache flushed to datafiles), the SCN is noted and the file headers (which store the SCN) are frozen.

Obviously, the SCN in a read only tablespace will not be up to date with the rest of the database. This is an exception that does not apply to other tablespaces.

You cannot perform any DML operations against a read-only tablespace. You can drop objects from it because a DROP updates the Data Dictionary which is not a read-only tablespace.

Read-only tablespaces only need to be backed up once, because they never change. It is still possible to back them up regularly, just pointless.

If RMAN is configured with BACKUP OPTIMIZATION ON, then it will only back up the tablespace to satisfy the retention policy.

If the status of a read-only tablespace is changed to read/write then it must be backed up immediately. Otherwise, RMAN will require all archive logs since the last backup (which can be months, since read only tablespaces may not be backed up consistently) if a restore is required.

Recovering from Loss of the Password File

Refresher: the password file is used to authenticate users when the db is not open (because it cannot access the Data Dictionary). The init parameter remote_login_passwordfile (if not set to none) specifies whether the password file is used to authenticate (regular) usernames with SYSDBA privileges or not. The password file is mainly used for authenticating SYSDBA over a network (sqlplus running on a client, connecting to a server).

The password file is a secondary method of authentication. OS authentication is always available.

RMAN does not backup the password file.

If the password file is damaged, then the instance will continue to function as normal, except that remote SYSDBA connections will not be possible. Startup will fail when the instance tries to read the damaged password file. The quick fix is to set the REMOTE_LOGIN_PASSWORD (static) parameter to NONE.

Another option is to recreate the password file:

orapwd file=<filename> password=<password> entries=<max_users>

Views & Tables

V$TABLESPACE

V$TEMPFILE

V$DATAFILE

V$LOG

V$LOGFILE

V$RECOVER_FILE

DBA_TABLESPACES

DBA_DATA_FILES

DBA_TEMP_FILES

DBA_USERS

DATABASE_PROPERTIES

DBA_SEGMENTS

Parameters

REMOTE_LOGIN_PASSWORD

Parallel Instance Recovery

Oracle RAC Instance Recovery.

1. All nodes available.

2. One or more RAC instances fail.

3. Node failure is detected by any one of the remaining instances.

4. Global Resource Directory(GRD) is reconfigured and distributed among surviving nodes.

5. The instance which first detected the failed instance, reads the failed instances redo logs to determine the logs which are needed to be recovered.

The above task is done by the SMON process of the instance that detected failure.

6. Until this time database activity is frozen, The SMON issues recovery requests for all the blocks that are needed for recovery. Once all the blocks are available, the other blocks which are not needed for recovery are available for normal processing.

www.csitportal.com

7. Oracle performs roll forward operation against the blocks that were modified by the failed instance but were not written to disk using redo log recorded transactions.

8. Once redo logs are applied, uncomitted transactions are rolled back using undo tablespace.

9. Database on the RAC in now fully available.