Database Startup and Shutdown

Table spaces

1. Introduction
2. Tablespaces and Data Files
3. Types of Tablespaces
4. Hierarchy
5. Space Management in Tablespaces
6. Creating Tablespaces

CONTROL_FILES

1. Introduction
2. When to create a control file
3. Control file contents
4. Provide file names to control files
5. Tables which holds control file information
6. How to identify control file location
7. Back up control files

ARCHIVELOG FILES

REDO LOGS

1. Introduction
2. Redo Log Contents
3. Redolog Components
4. How Oracle Database Writes to the Redo Log
5. Log switch & Log Sequence

DEMO -

- Practical on Table spaces
- Practical on Control files
- Practical on Redo log files

1. Introduction:

Starting Up a Database

- We startup an instance by mounting and opening the database. Doing so makes the database available for any valid user to connect to and perform data access operations.

Starting Up a Database

- You can start up a database instance with SQL*Plus.

Starting Up a Database Using SQL*Plus

- You can start a SQL*Plus session, connect to Oracle Database with administrator

Privileges, and then issue the STARTUP command.

```
SQL>Sqlplus '/as sysdba'
        Startup
```

- To start an instance, the database must read instance configuration parameters (the

Initialization parameters) from either a server parameter file (SPFILE) or a text

Initialization parameter file.

- When you issue the SQL*Plus STARTUP command, the database attempts to read the initialization parameters from an SPFILE. If it finds no SPFILE, it searches for a text initialization parameter file
- Default location for parameter file is $ORACLE_HOME/dbs
- In the default location, Oracle Database locates your initialization

Parameter files in the following order:

1. Spfile$ORACLE_SID.ora
2. spfile.ora
3. init$ORACLE_SID.ora

The first two filenames represent SPFILEs and the third represents a text initialization parameter file.

- Allocating the SGA
- Starting the background processes
- Opening the alertSID.log file and the trace files

2. Startup options

There are three modes available in startup namely
1)startup nomount
2)startup mount
3)Open

2.1. Startup nomount

An instance would be started in the NOMOUNT stage only during database creation or the re-creation of control files
At the time of startup nomountmode, it reads the parameter file which contain database parameters like

- Database name
- Control file location
- Diagnostic file location
- Processors value
- Memory required and
- Once the parameter file is read, it allocates memory for the database and starts the background processes of a database
- This does not allow access to the database and usually would be done only for database creation or the re-creationof control files.

Syntax:
SQL>startup nomount

2.2. Startup mount

- In startup mount, the instance reads the control file which contains the location of data files and Redo log files with checkpoint information
- This state allows for certain DBA activities, but does not allow general access to the database. For example
- Enabling and Disabling archive mode.
- Performing full database recovery

Mounting a database includes the following tasks:

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

Syntax:

SQL>startup mount

If currently in nomount mode, then use

SQL>alter database mount

Restrict mount

- We will start the instance in restrict mode so that the instance is only available to administrative user. Use this mode of instance startup when
- Perform an export or import of data
- Perform a data load (with SQL*Loader)
- Temporarily prevent typical users from using data
- Perform certain migration or upgrade operations
- Drop the Database

Syntax:

SQL>startup restrict mount
To disable restrict mode
SQL>alter system disable restricted session;

2.3. Startup Open

In open mode, the data files and redo logs are opened and the database is ready for user access.

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

The Oracle server verifies that all the data files and online redo logfiles can be opened and checks the consistency of the database. If necessary, the SMON Background process initiates instance recovery.

Syntax:
SQL>startup

If currently in other mode, then use
SQL>alter database open

Note: To verify the Database mode, then use
SQL>select open_mode, name from v$database

3. Shutdown

There are four modes in shutdown to bring the database down

3.1. Shutdown Normal

- No recovery is required
- It will shutdown only when all the transactions are committed and restored
- Waits for user to get disconnect
- New transactions can be taken

```
SQL>shutdown
```

- In general, this mode will not be used

3.2. Shutdown immediate

- It doesn't require recovery
- New connections will not be accepted
- Old or existing connections will get disconnect
- The transaction that is committed is written into data files
- It is the most useful command used in general to shutdown the client database which brings the database down immediately
- It dismounts and closes the opened datafiles

```
SQL>shutdown immediate
```

3.3. Shutdown abort

- Requires recovery in the next instance startup
- User connections are terminated immediately
- Datafiles remain open
- The committed data will not be written into datafiles
- New connections will not be accepted

```
SQL>shutdown abort
```

3.4. Shutdown transactional

- New connections will not be accepted
- Does not require instance recovery in the next startup
- Data files get closed after shutdown
- Once the transaction is committed, it writes the data into datafiles and will disconnect the user

4. Parameter File

- This file contains parameters that will define the characteristics of database.
- It is a text file and will be in the form of init<SID>.ora and resides in ORACLE_HOME/dbs
- Parameters are divided into two types

(a) Static parameters- the value of these parameters cannot be changes when the database is up and running

(b) Dynamic parameters- the value for these parameters can be changed even when database is up and running

- Pfile can be edited using vi editor

➢ If both Pfile and Spfile is lost

There are two ways in which we can create a new pfile

(a) As we know that when we start database it will check for the initialisation parameters either from pfile or spfile. Those details will be shown in alert log which is located in dump_dest.we can build a new pfileusing those parameters, from that we can create a spfile

(b) We can copy a parameter file from another instance and edit as per our requirement

➢ if pfile is lost

As at the time of database startup,the instance reads the spfile.
Connect as sysdba
Sqlplus '/as sysdba'
Create pfile from spfile

➢ Edit a pfile

Whenever we want to edit the pfile, first we have to shutdown the database and modify the parameters which need to be modified and then startup the database using the pfile
Startup nomountpfile ='$ORACLE_HOME/dbs/init<SID>.ora

5. Server parameter File

- The format of spfile is “spfileSID.ora”
- Spfile is a binary file which cannot be edited using vi editor.
- The default location of spfile is ORACLE_HOME/dbs

To verify which parameter file being used by database
SQL>show parameter spfile

Note: If it is spfile, then the value will be displayed otherwise the value will not display

- The default parameter file being read at the time of startup nomount is spfile

For verifying the parameter to change dynamically or static:
SQL>select issys_modifiable, name from v$parameter where name like ‘parametername’
If issys_modifiable is immediate then change the parameter value dynamically by using spfile

For example:
SQL>select issys_modifiable, name from v$parameter where name like ‘undo_retention’;
Here, issys_modifiable is IMMEDIATE, so use

SQL>alter system set undo_retention=1000 scope=both

- If scope=both, then the value of the parameter reflects in current memory and also in the next startup.
- If scope=memory, then the current memory will effect but in next startup, it uses the old value from spfile
- If scope=spfile, then the current memory will not get affected, but in the next startup the effect appears in spfile

Select issys_modifiable,name from v$parameter where name like ‘undo_management’;
Here, issys_modifiable is FALSE

So, create pfile from spfile;
Shutdown immediate

Go to $ORACLE_HOME/dbs
Viinitp<SID>.ora
Now, change the parameter value in pfile
Startup the database using pfile
Startuppfile= ‘$ORACLE_HOME/dbs/init<SID>.ora’

Create spfile from pfile
Shutdown immediate
Startup
Show parameter undo_retention

6. Password File

- This file contain sydba password and will be used if any user with sysdba permission is trying to connect database
- It will be in the form of orapw<SID> and resides in ORACLE_HOME/dbs
- If we forgot password of sys user or lost password file, it can be created using ORAPWD utility

```
Orapwd file=orapwtest password=oracle entries=5 force=y entries=10
```

Force =Y will allow to overwrite the password file

Entries=10 represents the number of users that can use this password file or to how many members we can assign sysdbapriviliges

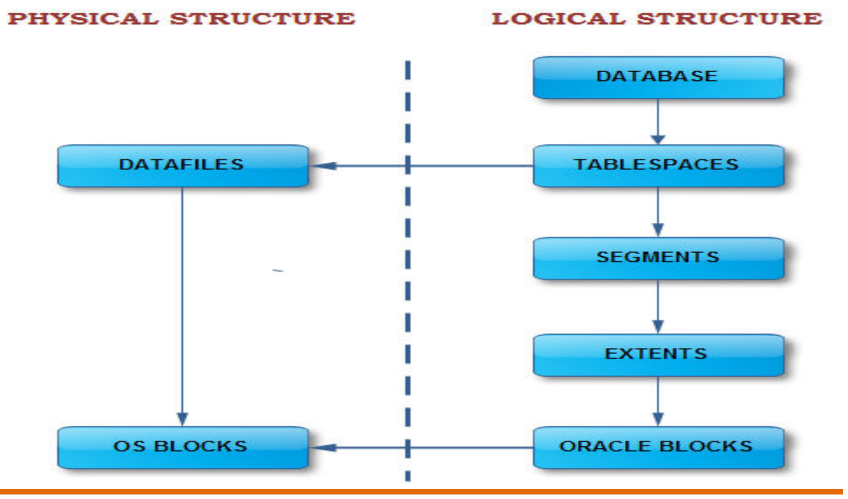
Password=We have to give the system password

- We can only create a passwordfile only if remote_login_password=EXCLUSIVE

Table space Management

1. Introduction

- A tablespace is a logical storage unit where multiple tables are stored
- A database can contain multiple tablespaces
- Tablespaces Consist of one or more data files
- Data files can belong to only one tablespace and one database



2. Tablespaces and Data Files:

Databases, tablespaces, and data files are closely related, but they have important differences:

• An Oracle database consists of one or more logical storage units called tablespaces, which collectively store all of the database's data.

• Each tablespace in an Oracle database consists of one or more files called data files, which Are physical structures that conform with the operating system in which Oracle is running.

• A database's data is collectively stored in the data files that constitute each tablespace of the database. For example, the simplest Oracle database would have one tablespace and one data file. Another database can have three tablespaces, each consisting of two data files (for a total of six data files).

3. Types of Tablespaces

The DBA creates tablespaces for increased control and ease of maintenance. The Oracle server perceives two types of tablespaces: SYSTEM and all others

1) SYSTEM
2) Non-System

3.1. System Tablespaces:

• Created with the database

• Required in all databases

• Contains the data dictionary, including stored program units

• Contains the SYSTEM undo segment

• Should not contain user data, although it is allowed

The primary tablespace in any database is the SYSTEM tablespace, which contains information basic to the functioning of the database server, such as the data dictionary and the system rollback segment. The SYSTEM tablespace is the first tablespace created at database creation. It is managed as any other tablespace, but requires a higher level of privilege and is restricted in some ways. For example, you cannot rename or drop the SYSTEM tablespace or take it offline.

The SYSAUX tablespace, which acts as an auxiliary tablespace to the SYSTEM tablespace, is also always created when you create a database. It contains information about and the schemas used by various Oracle products and features, so that those products do not require their own tablespaces. As for the SYSTEM tablespace, management of the SYSAUX tablespace requires a higher level of security and youcannot rename or drop it.

Note: System Table space holds system related data dictionary information owned by sys user.

Eg: dba_users, dba_data_files, v$parameter, dba_objects, all_objects

3.2. Non-SYSTEM tablespaces

• Enable more flexibility in database administration

• Separate undo, temporary, application data, and application index segments

• Separate data by backup requirements

• Separate dynamic and static data

• Control the amount of space allocated to the user's objects

4. Hierarchy

4.1. Segments:

Segments are made up of multiple extents. One or more segments make up a tablespace

It is set of extents allocated for specific data structure (like table or index).

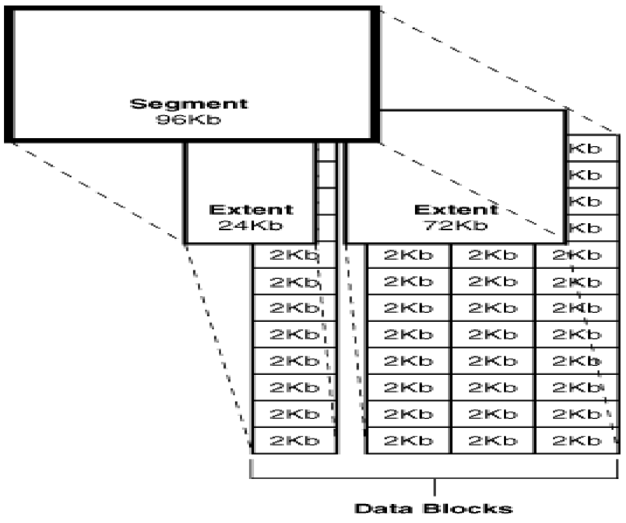
SL.NO	**TYPES**	**PURPOSE**
1	Data segment	Stores user data within the database.
2	Rollback segment	Stores rollback information used when data must be rolled back.
3	Index segment	Stores indexes.
4	Temporary segment	Created when a SQL statement requires a temporary work area such as during sorting of data.

4.2. Extent:

- An extent is a logical unit of database storage space allocation made up of contiguous data blocks.
- By default, the database allocates an initial extent for a data segment when the segment is created.

4.3. Oracle Blocks:

- At the finest level of granularity, Oracle stores data in data blocks (also called logical blocks, Oracle blocks, or pages).
- One data block corresponds to a specific number of bytes of physical database space on disk.



5. Space Management in Tablespaces

5.1. Locally managed tablespaces:

The extents are managed within the tablespace via bitmaps.

Each bit in the bitmap corresponds to a block or a group of blocks. When an extent is allocated or freed for reuse, the Oracle server changes the bitmap values to show the new status of the blocks. Locally managed is the default beginning with Oracle9i.

• Reduced contention on data dictionary tables

• No undo generated when space allocation or deallocation occurs

• No coalescing required

```
CREATE TABLESPACE userdata DATAFILE '/u01/oradata/userdata01.dbf' SIZE 500M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

- The EXTENT MANAGEMENT clause can be used in various CREATE commands:

• For a permanent tablespace, you can specify EXTENT MANAGEMENT LOCAL in the CREATE TABLESPACE command.

Note: Prior to Oracle9i Database Release 2, the SYSTEM tablespace was not locally managed.

• For a temporary tablespace, you can specify EXTENT MANAGEMENT LOCAL in the CREATE TEMPORARY TABLESPACE command

5.1.1. Advantages of Locally Managed Tablespaces

Locally managed tablespaces have the following advantages over dictionary-managed tablespaces:

• Local management avoids recursive space management operations. This can occur in Dictionary-managed tablespaces if consuming or releasing space in an extent results in another operation that consumes or releases space in an undo segment or data dictionary table.

• Because locally managed tablespaces do not record free space in data dictionary tables, they reduce contention on these tables.

• Local management of extents automatically tracks adjacent free space, eliminating the need to coalesce free extents.

• The sizes of extents that are managed locally can be determined automatically by the system.

• Changes to the extent bitmaps do not generate undo information because they do not update tables in the data dictionary (except for special cases such as tablespace quota information).

5.2. Dictionary-managed tablespaces:

The extents are managed by the data dictionary. The Oracle server updates the appropriate tables in the data dictionary whenever an extent is allocated or deallocated.

• Extents are managed in the data dictionary.

• Each segment stored in the tablespace can have a different storage clause.

• Coalescing is required.

CREATE TABLESPACE userdata DATAFILE '/u01/oradata/userdata01.dbf' SIZE 500M EXTENT MANAGEMENT DICTIONARY DEFAULT STORAGE (initial 1M NEXT 1M PCTINCREASE 0);

❖ Segments in dictionary-managed tablespaces can have a customized storage. This storage is more flexible than locally managed tablespaces but much less efficient.

Migrate a dictionary managed SYSTEM tablespace to locally manage:

DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL ('SYSTEM');

6. Creating Tablespaces

You create a tablespace with the CREATE TABLESPACE command:

```
CREATE TABLESPACE tablespace
[DATAFILE clause]
[MINIMUM EXTENT integer[K|M]]
[BLOCKSIZE integer [K]]
[LOGGING|NOLOGGING]
[DEFAULT storage_clause ]
[ONLINE|OFFLINE]
[PERMANENT|TEMPORARY]
[extent_management_clause]
[segment_management_clause]
```

Where:

Tablespace: This is the name of the tablespace to be created.

DATAFILE: This specifies the data file or data files that make up the tablespace.

MINIMUM EXTENT: This ensures that every used extent size in the tablespace is a multiple of the integer. Use K or M to specify this size in kilobytes or megabytes.

BLOCKSIZE: BLOCKSIZE: Specifies a nonstandard block size for the tablespace. In order to specify this clause, you must have the DB_CACHE_SIZE and at least one DB_nK_CACHE_SIZEparameter set, and the integer you specify in this clause must correspond with the setting of one DB_nK_CACHE_SIZEparameter setting.

LOGGING: This specifies that, by default, all tables, indexes, and partitions within the tablespace have all changes written to online redo log files. LOGGING is the default.

NOLOGGING: This specifies that, by default, all tables, indexes, and partitions within the tablespace do not have all changes written to online redo log files. NOLOGGING affects only some DML and DDL commands, for example, direct loads.

DEFAULT: DEFAULT: Specifies the default storage parameters for all objects created in the tablespace creation.

OFFLINE: This makes the tablespace unavailable immediately after creation.

PERMANENT: This specifies that the tablespace can be used to hold permanent objects.

TEMPORARY: This specifies that the tablespace be used only to hold temporary objects; for example, segments used by implicit sorts caused by an ORDER BY clause. It cannot specify

EXTENT MANAGEMENT LOCAL or the BLOCKSIZE clause.

extent_management_clause: This clause specifies how the extents of the tablespace are managed. This clause is discussed in a subsequent section of this lesson.

segment_management_clause: This is relevant only for permanent, locally managed tablespaces. It lets you specify whether Oracle should track the used and free space in the segments in the tablespace using free lists or bitmaps.

datafile_clause:== filename [SIZE integer[K|M] [REUSE]

[autoextend_clause]

filename: This is the name of a data file in the tablespace.

SIZE: This specifies the size of the file. Use K or M to specify the size in kilobytes or megabytes.

REUSE: This allows the Oracle server to reuse an existing file.

Autoextend_clause: This clause enables or disables automatic extension of the data file.

NEXT: This specifies the size in bytes of the next increment of disk space to be allocated automatically when more extents are required.

<u>BIGFILE TABLESPACE</u>

1. For managing the datafiles in VLDB, oracle introduced bigfile tablespace in 10g
2. Bigfile tablespace's datafiles can grow into terabytes based on the block size. For example, for a 8KB block size a single file can grow till 4TB
3. Bigfile tablespaces should be created only when we have stripping and mirroring implemented at storage level in real time
4. We can't add another datafile to a bigfile tablespace until it reaches max value
5. Bigfile tablespaces can be created only as LMT and with ASSM

Note: Either in LMT or DMT, ASSM once defined cannot be changed

ARCHIVELOG FILES

INTRODUCTION

Archive logs are known as offline redo log files. Oracle database allows the redo logs to be saved in particular destination after the redo log groups are filled or switched the groups which are active are written to this destination. This destination is called archive logs and the process is called archiving. This process is only possible when the database is in archive log mode and taken care by a process called archiever (It is an optional background process). Archive logs include data of redo log files, log sequence number. There are two modes of archiving automatic and manual. By default automatic archive log mode is taken as default while changing to archive log mode if the mode is not specified. The primary value of archive logs is recovery.

In manual archiving, the archiving of redo logs is done manually instead of automatically. It is done by using the following command

```
SQL>alter system archive log all
```

In automatic archiving, the archiving of redo logs is done automatically. We can manually archive while in automatic mode to rearchive inactive redo groups to another location. It will throw an error if the redo group we have to rearchive is already overwritten.

1. Archive log files are offline copies for online redolog files and are required to recover the database if we have old backup
2. Archivelog generation can be of two ways – manual and automatic. It is always preferred to use automatic method as DBA's cannot be dedicated to perform manual archiving
3. The following are parameters that are used for archivelog mode with their description
    a. LOG_ARCHIVE_START – it will enable automatic archiving and useful only till 9i (deprecated in 10g)
    b. LOG_ARCHIVE_TRACE – it is used to generate a trace file to know how ARCHn process working
    c. LOG_ARCHIVE_MIN_SUCCEEDED_DEST – defines min destinations to which ARCHn process should complete archiving by the time LGWR starts writing to online redolog file
    d. LOG_ARCHIVE_MAX_PROCESSES – will start multiple ARCH processes and helpful in faster writing
    e. LOG_ARCHIVE_LOCAL_FIRST – if enabled, ARCHn process will first generate archive in local machine and then in remote machine. It is used in case of dataguard setup
    f. LOG_ARCHIVE_FORMAT – defines the archive log file format
    g. LOG_ARCHIVE_DUPLEX_DEST – if want to archive in only 2 locations, we should use this
    h. LOG_ARCHIVE_DEST_1...10 – if want to archive to more than 2, we should enable this

i. LOG_ARCHIVE_DEST_STATE_1...10 – to enable / disable archive locations
j. LOG_ARCHIVE_CONFIG – it enables / disables sending redologs to remote location. Used in dataguard environment

4. When we want to multiplex into only 2 locations, from 10g we should use LOG_ARCHIVE_DEST and LOG_ARCHIVE_DUPLEX_DEST parameters
5. The default location for archivelogs in 10g is Flash Recovery Area (FRA). The archives in this location are deleted when a space pressure arised. The location and size of FRA can be known using DB_RECOVERY_FILE_DEST and DB_RECOVERY_FILE_DEST_SIZE parameters respectively
6. To disable archivelog generation into FRA, we shouldn't use LOG_ARCHIVE_DEST, but should use LOG_ARCHIVE_DEST_1

Note: archivelogs can also be deleted based on RMAN deletion policy

TABLES RELATED TO ARCHIVE LOGS

- V$DATABASE:- Shows if the database is in ARCHIVELOG or NOARCHIVELOG mode and if MANUAL (archiving mode) has been specified.
- V$ARCHIVED_LOG:- Displays historical archived log information from the control file. If you use a recovery catalog, the RC_ARCHIVED_LOG view contains similar information.
- V$ARCHIVE_DEST:-Describes the current instance, all archive destinations, and the current value, mode, and status of these destinations.
- V$ARCHIVE_PROCESSES:- Displays information about the state of the various archive processes for an instance.
- V$BACKUP_REDOLOG:-Contains information about any backups of archived logs. If you use a recovery catalog, the RC_BACKUP_REDOLOG contains similar information.
- V$LOG:-Displays all redo log groups for the database and indicates which need to be archived.
- V$LOG_HISTORY:-Contains log history information such as which logs have been archived and the SCN range for each archived log.

The SQL*Plus command ARCHIVE LOG LIST displays archiving information for the connected instance.

```
SQL> archive log list;
```

It gives us information whether archive log mode is enabled or not, automatic archiving is enabled or not, archive destination, current log sequence, next log sequence to archive, oldest online log sequence.

COMMANDS

<u>Follow the below step to change to archive log mode.</u>

```
SQL> shut immediate;

SQL> startup mount;

SQL> alter database archivelog;

SQL> alter database open;

SQL> select name, log_mode from v$database;

SQL> alter system switch logfile;
```

<u>Follow the below step to change to no archive log mode.</u>

```
Shutdown immediate;

Startup mount;

Alter system archive log stop;

Alter database noarchivelog;

Archive log list;

Alter database open;
```

Note: When we enable archivelog mode using above method, the archives will be generated by default in Flash Recovery Area (FRA). It is the location where files required for recovery exist and introduced in 10g

# To change the archive destination from FRA to customized location

1. SQL> alter system set log_archive_dest_1='location=/u03/archives' scope=spfile;
2. SQL> shutdown immediate
3. SQL> startup

CONTROL_FILES

1. Introduction

A control file is the **binary file** which contains the file structure of a database. It includes

1) Database name
2) Names and locations of associated datafiles and redo log files
3) The timestamp of the database creation
4) The current log sequence number
5) Checkpoint information
6) Current online redo log file sequence number
7) Begin and end of undo segments
8) Back up information
9) Redo log archive information

NOTE:

- Control file must be there to mount a data base; otherwise it is difficult to do recovery.
- Control file should be created while database creation.
- At least one control file should be there for each data base.
- Only oracle server can do update in control file.
- Before database open, control file is read to check the validity of database.

2. Create a control file

- First time when we are creating a new database using CREATE DATABASE, control files can be created. The file names will be specified in that statement. Those names are used as initialization parameters by database.
- In case if we want to change the existing DB_NAME.
- All control files for the database damaged permanently and no backup of control file is available.

Creating Additional Copies, Renaming, and Relocating Control Files:

- We can create an additional control file copy for multiplexing by copying an existing control file to a new location and adding the file name to the list of control files.
- We also can rename an existing control file by copying the file to its new name or location, and changing the file name in the control file list

3. Control file contents:

The control file has two sections.

- Reusable
- Not reusable

Reusable sections store the RMAN information like names of back up data files and back up online redo log file.

4. Provide file names to control files:

You specify control file names using the CONTROL_FILES parameter in the pfile/spfile.

If you do not specify files for CONTROL_FILES before database creation:

- If you are not using Oracle-managed files, then the database creates a control file and uses a default filename. The default name is operating system specific.
- If you are using OMF, then the parameters you set to enable that feature determine the name and location of the control files.
- If you are using Automatic Storage Management, you can place incomplete ASM filenames in the DB_CREATE_FILE_DEST and DB_RECOVERY_FILE_DEST initialization parameters. ASM then automatically creates control files in the appropriate places.
- Names are uniquely generated and displayed in the alertSID.log

5. Tables which holds control file information:

The following views display information about control files:

View	**Description**
V$DATABASE	Displays database information from the control file
V$CONTROLFILE	Lists the names of control files
V$CONTROLFILE_RECORD_SECTION	Displays information about control file record sections
V$PARAMETER	Displays the names of control files as specified in the CONTROL_FILES initialization parameter

6. Identify control file location

We can check for control file parameter in pfile(initSID.ora) , which has the location of control file. Or else,

Connect to sqlplus as sysdba

```
[oravis@ebs12trn ~]$ sqlplus / as sysdba
```

```
SQL>show parameter control_files;
```

7. Back up control files:

The control file can be backed up in the case when we are doing any structural modifications in database

- Adding, dropping, or renaming datafiles
- Adding or dropping a tablespace, or altering the read/write state of the tablespace
- Adding or dropping redo log files or groups

NOTE: RMAN backup can take the back up of control files.

NOTE: Control files can be backed up using below commands.

- Back up the control file to a binary file (duplicate of existing control file) using the following statement:

```
ALTER DATABASE BACKUP CONTROLFILE TO trace as '/location/ctrl.bkp';
```

- Produce SQL statements that can later be used to re-create your control file:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

REDO LOGS

1. Introduction

The most crucial structure for recovery operations is the redo log.

Every instance of an Oracle Database has an associated redo log to protect the database in case of an instance failure

The redo log for each database instance is also referred to as a redo thread

The initial set of online redo log file groups and members are created during the database
Creation.

The following parameters limit the number of online redo log files:

The MAXLOGFILES parameter in the CREATE DATABASE command specifies the absolute maximum of online redo log file groups.

The MAXLOGMEMBERS parameter used in the CREATE DATABASE command
Determines the maximum number of members per group.

The maximum and default value for MAXLOGFILES and MAXLOGMEMBERS is dependent on your operating system.

2. Redo Log Contents

Redo log files are filled with redo records, also called a redo entry, 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

LGWR writes the transaction redo records from the redo log buffer of the SGA to a redo log file, and assigns a system change number (SCN)

Only when all redo records associated with a given transaction are safely on disk the user process notified that the transaction has been committed.

3. Redolog Components:

3.1. Redo Log File Groups:

A set of identical copies of online redo log files is called an online redo log file group.

The LGWR background process concurrently writes the same information to all online redo log files in a group.

The Oracle server needs a minimum of two online redo log file groups for the normal operation of a database.

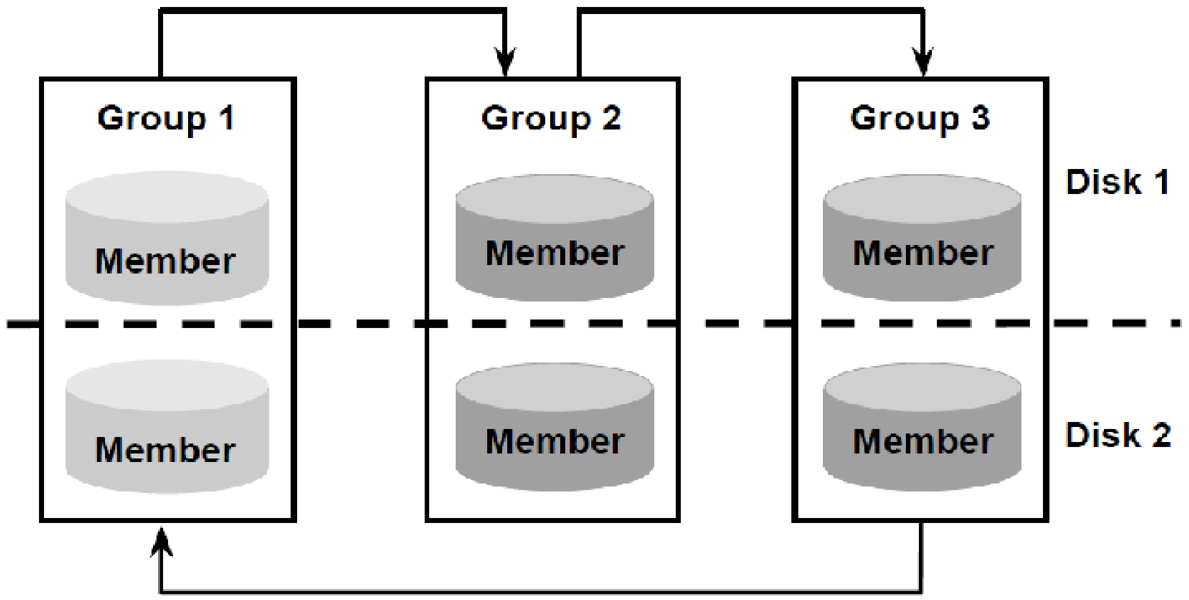
3.2. Redo Log File Members

Each online redo log file in a group is called a member.

Each member in a group has identical log sequence numbers and are of the same size.

4. Oracle Database Writes to the Redo Log

LGWR writes to redo log files in a circular fashion. When the current redo log file fills,LGWR begins writing to the next available redo log file. When the last available redo log file is filled, LGWR returns to the first redo log file and writes to it.



The redo log file that LGWR is actively writing to is called the current redo log files.

Redo log files that are required for instance recovery are called active redo log files.

Redo log files that are no longer required for instance recovery are called inactive redo log files.

5. Log switch & Log Sequence

A log switch is the point at which the database stops writing to one redo log file and begins writing to another.

Normally, a log switch occurs when the current redo log file is completely filled and writing must continue to the next redo log file.

Log switch can occur at regular interval by setting ARCHIVE_LAG_TARGET parameter in parameter file.

ex:-ARCHIVE_LAG_TARGET = 1800

Oracle Database assigns each redo log file a new log sequence number every time a log switch occurs and LGWR begins writing to it

During crash, instance, or media recovery, the database properly applies redo log files in ascending order by using the log sequence number

By default each database will have five tablespaces in oracle 10g.

1. System : Tablespace-contains data dictionary of database
2. Sysaux : Contains database statistics
3. Undo : Conatins Pre Image data
4. Temporary : Temporary oprations are performed in this hz if pga is not enough
5. Users : Default tablespace for all DB users / Application schemas

Table Space Demo

1. Creating table space with auto extend clause:

Sql> create tablespace <name> datafile '/u01/home/prod/tsb.dbf' size 10m autoextend on maxsize 100m default storage (next 10m);

Tablespace created

Note:

- Here initially tablespace is created with 10mb.
- And size of tablespace is increased by 10mb upto 100mb. i.e if 10mb is full then size of tablespace is extended to 20mb.

Disable autoextend:

- SQL> alter database datafile '/u01/db/oraprod/oradata/prod/tem01.dbf' autoextend off;

2. Create a Tablespace:

- While creating a tablespace first check the location of the tablespace.

SQL> select file_name from dba_data_files;

- SQL> create tablespace tem datafile '/u01/db/oraprod/oradata/prod/tem01.dbf' size 50m;
- Tablespace created.
- Check whether datafile is created or not.
- SQL> select file_name from dba_data_files;

NOTE: Here we can see the data file which we had created.

3. Adding a datafile to a tablespace

- SQL>alter tablespace tem add datafile '/u01/db/oraprod/oradata/prod/tem02.dbf' size 50m;
- Now check the data file is added to the tablespace or not.
  Sql>select file_name from dba_data_files;

Deleting a datafile from a tablespace

Sql> alter tablespace tem drop datafile '/u01/db/oraprod/oradata/prod/temp02.dbf';

- Now check the data file is deleted from the tablespace or not.
  Sql>select file_name from dba_data_files;

4. Droping a tablespace

Sql> drop tablespace hz;

Note: It will drop tablespace logically at database level.

Sql> drop tablespace hz including contents and datafiles;

Note: It will drop tablespace

logically(database level) and physically(o/s level)

5. Reusing orphan datafile

Sql> create tablespace <name> datafile '/u01/user18/demo/data/hz01.dbf' reuse;

6. Resize a Datafile

First check size of the tablespace we had created.

SQL> select file_name, bytes/1024/1024 from dba_data_files where tablespace_name like '%TEM%';

Note: Here we observe the size of the datafile is 50 mb now we try to change the size of datafile.

SQL> alter database datafile '/u01/db/oraprod/oradata/prod/tem01.dbf' resize 60m;

- Now check the size has been changed or not.

SQL> select file_name, bytes/1024/1024 from dba_data_files where tablespace_name like '%TEM%';

7. Making a tablespace as read only

```
Sql> alter tablespace hz read only;

Sql> select tablespace_name,status from dba_tablespaces;

Sql> alter tablespace hz read write;
```

8. Making a tablespace offline

```
Sql> alter tablespace hz offline;
```

Note: Users can not access this tablespace in this state.

```
Sql> alter tablespace hz online;
```

9. Renaming of tablespace

```
Sql> alter tablespace hz rename to hz1;
```

10. Renaming a datafile in tablespace

Step1:make tablespace offline

```
Sql> alter tablespace hz offline;
```

Step2:At os level rename the datafile

```
[oracle@.....~]$ cd /u01/user18/demo/data/

[oracle@....data~]$ mv hz01.dbf  hz03.dbf
```

Step3: Update the controlfile for this datafile.

```
Sql> alter database rename file ‘/u01/user18/demo/data/hz01.dbf’ to

        ‘/u01/user18/demo/data/hz03.dbf’;
```

Step4: Online the tablespace

```
Sql> alter tablespace hz online;

Sql> select tablespace_name, file_name from dba_data_files;
```

11.Relocating a datafile in tablespace

Step1:make tablespace offline

```
Sql> alter tablespace hz offline;
```

Step2: at os level rename the datafile

```
$cd /u01/user18/demo/data/

$mv hz03.dbf  ../hz03.dbf
```

Step3: update the controlfile for this datafile.

```
Sql> alter database rename file '/u01/user18/demo/data/hz03.dbf' to

        '/u01/user18/demo/hz03.dbf';
```

Step4: online the tablespace

```
Sql> alter tablespace hz online;

Sql>Select tablespace_name,file_name from dba_data_files;
```

12.Moving table from on tablespace to another tablespace

```
Sql> alter table emp move tablespace hz;
```

13.Moving index from on tablespace to another tablespace

```
Sql> alter index emp_indx rebuild tablespace hz;
```

14.To check database size

```
Sql>select  sum(bytes)/1024/1024 "size in MB" from dba_data_files;
```

15.To check free space in database

```
Sql>select sum(bytes)/1024/1024 from dba_free_space;
```

16. To find the block size

```
SQL> show parameter db_block_size;
```

17. To identify total number of data files.

```
SQL> show parameter db_files;
```

18. To find the existing table spaces.

```
SQL> desc dba_tablespaces;

SQL> select tablespace_name from dba_tablespaces;
```

19. To find out existing data files.

```
SQL> desc dba_data_files;

SQL> select file_name from dba_data_files;
```

20. To know the default tablespace set for a database.

```
SQL> desc database_properties;

SQL>select * from database_properties where property_name
like ‘%TABLESPACE%’;
```

21. Views:

- V$tablespace
- V$datafile
- Dba_tablespaces
- User_tablespaces
- Dba_data_files
- Dba_segments
- Dba_extents
- sm$ts_free
- Sm$ts_used
- sm$ts_avail

Control file Practical

1. Manual creation of control file:

We can create control file of database manually by using CREATE CONTROLFILE statement. This will be used mostly in case of creating a new database.

```
CREATE CONTROLFILE

  SET DATABASE prod

  LOGFILE GROUP 1 ('/u01/oracle/prod/redo01_01.log',

              '/u01/oracle/prod/redo01_02.log'),

        GROUP 2 ('/u01/oracle/prod/redo02_01.log',

              '/u01/oracle/prod/redo02_02.log'),

        GROUP 3 ('/u01/oracle/prod/redo03_01.log',

              '/u01/oracle/prod/redo03_02.log')

  RESETLOGS

  DATAFILE '/u01/oracle/prod/system01.dbf' SIZE 3M,

        '/u01/oracle/prod/rbs01.dbs' SIZE 5M,

        '/u01/oracle/prod/users01.dbs' SIZE 5M,

        '/u01/oracle/prod/temp01.dbs' SIZE 5M

  MAXLOGFILES 50

  MAXLOGMEMBERS 3

  MAXLOGHISTORY 400

  MAXDATAFILES 200

  MAXINSTANCES 6

  ARCHIVELOG;
```

2. Controlfile recovery from trace:

We can recover the controlfile from trace when all controlfiles are lost or damaged and it is possible only when we backup the controlfile to trace.

```
Sql>  alter database backup controlfile to trace;

          Database altered

          Sql>
```

Step1: Now controlfile is available at trace. So we find the tracefile location.

```
          Sql> select value from v$diag_info;
```

So here the last row specifies the controlfile backup. i.e

/u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_3098.trc

Step2: Copy that file to some other location

```
          Sql> shut immediate;

          [oracle@.....      ~]$cp      /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_3098.trc
/u01/app/oracle/orcl_ora_3098.sql
```

Here file should be saved with .sql extension.

Step3: Now open orcl_ora_3098.sql in vi editor.

```
[oracle@....~]$ cd /u01/app/oracle

[oracle@..... oracle ~]$ vi orcl_ora_3098.sql

STARTUP NOMOUNT

CREATE CONTROLFILE REUSE DATABASE "ORCL" NORESETLOGS  ARCHIVELOG

    MAXLOGFILES 16

    MAXLOGMEMBERS 3

    MAXDATAFILES 100

    MAXINSTANCES 8
```

```
   MAXLOGHISTORY 292
LOGFILE
  GROUP 1 '/u01/app/oracle/oradata/orcl/redo01a.log'  SIZE 10M BLOCKSIZE 512,
  GROUP 2 '/u01/app/oracle/oradata/orcl/redo02.log'  SIZE 50M BLOCKSIZE 512,
  GROUP 3 '/u01/app/oracle/red003b.log'  SIZE 10M BLOCKSIZE 512,
  GROUP 4 '/u01/app/oracle/flash_recovery_area/ORCL/onlinelog/o1_mf_4_b8jbn6bh_.log'  SIZE 100M
BLOCKSIZE 512,
  GROUP 5 '/u01/app/oracle/redo05b.log'  SIZE 10M BLOCKSIZE 512
-- STANDBY LOGFILE
DATAFILE
  '/u01/app/oracle/oradata/orcl/system01.dbf',
  '/u01/app/oracle/oradata/orcl/sysaux01.dbf',
  '/u01/app/oracle/oradata/orcl/undotbs01.dbf',
  '/u01/app/oracle/oradata/orcl/users01.dbf',
  '/u01/app/oracle/oradata/orcl/example01.dbf',
  '/u01/app/oracle/es.dbf',
  '/u01/app/oracle/es1.dbf',
  '/u01/app/oracle/oradata/ee.dbf'
CHARACTER SET WE8MSWIN1252
;
```

Note: Here keep the scriptshown above in the editor and delete remaining lines of data (dgg – to delete lines above the cursor, dG – to delete lines below the cursor) and save the file.

```
[oracle@......oracle ~]$ :wq
[oracle@......oracle ~]$
```

Step4: Now startup the database to nomount state.

```
[oracle@......oracle ~]$ sqlplus / as sysdba

SQL>  startup nomount

ORACLE instance started.

          Total System Global Area  849530880 bytes

          Fixed Size               1339824 bytes

          Variable Size            545263184 bytes

          Database Buffers         297795584 bytes

          Redo Buffers             5132288 bytes

          Sql>
```

- **Now execute the script which is saved in .sql format**

```
Sql> @ /u01/app/oracle/orcl_ora_3098.sql

Controlfile created

Sql>
```

- **Here database is automatically mounted.**

```
Sql> alter databae open resetlogs;

Database altered
```

Open reset locks will reset the log sequence and current sequence number starts with new sequence.

3. Multiplexing of controlfiles:

If we lost control file, we need to do recovery to stable the database. In order to avoid that, better to keep identical copy of control file in different location, which is meant to be multiplexing of control file.

Control file can be multiplexed up to eight times by:

3.1. Multiplexing at the time of database creation:

Step1: Creating multiple control files when the database is created by including the control file names and full path in the initialization parameter file:

```
Control_files: /location/ctrl01.ctl,
                                ----
                                ----
                             /location/ctrl08.dbf
```

Step2: Adding a control file after database created.

3.2. Multiplexing after database creation:

- **<u>Steps to create control file using pfile:</u>**

  - Shutdown database;
    ```
    Sqlplus / as sysdba
    Sql> shut immediate;
    Database closed.
    Database dismounted.
    ORACLE instance shut down
    Sql>exit
    [oracle@......~]$
    ```

  - Backup of pfile
    ```
    [oracle@.... ~]cp $ORACLE_HOME/dbs/initSID.ora $ORACLE_HOME/dbs/initSID.ora_old
    ```

  - Modify pfile
    ```
    [oracle@.... ~]$ cd $ORACLE_HOME/dbs
    [oracle@.... ~]$ viinitSID.ora
    ```

    ```
    Control_files= /location1/ctrl01.ctl,/location2/ctrl02.ctl(new entry)
    ```

    Save the pfile.
  - Physical creation of file in new location

    ```
    [oracle@.... ~]$ cp /location/ctrl01.ctl /location/ctrl02.ctl
    ```

- Start database.

```
Sqlplus / as sysdba
Sql> create spfile from pfile;
Sql>startup;
```

- **<u>Steps to create control file using spfile:</u>**

- Change the control file_parameter.

```
[oracle@.... ~]$ sqlplus / as sysdba
Sql>Alter system set control_files='/location/ctrl01.ctl','
                                        '/location/ctrl02.ctl'(new entry)scope=both;
```

- Stop the database.

```
          Sqlplus / as sysdba
          Sql> shut immediate;
          Database closed.
          Database dismounted.
          ORACLE instance shut down
          Sql>exit
          [oracle@......~]$
```

- Create a copy of the control file in new location

```
[oracle@.... ~]$ cp /location/ctrl01.ctl /location/ctrl02.ctl
```

- Start the database.

```
[oracle@.....~]$ sqlplus / as sysdba
Sql>startup;
```

4. Drop a controlfile:

We can drop a control file from database, if mount point or control file itself are no longer valid.

But before dropping control file, we should make sure that our environment must have one more back up of control file in addition to the original control file.

Step1: Shut down the database.

```
Sql> shut immediate;

          Database closed.
          Database dismounted.
          ORACLE instance shut down
          Sql>exit

[oracle@......~]$
```

Step2: Edit the CONTROL_FILES parameter in the database initialization parameter file to delete the old control file name.

```
[oracle@....~]$ vi $ORACLE_HOME/dbs/init<sid>.ora
CONTROL_FILES=/location1/ctrl01.ctl, /location2/ctrl02.ctl
```

- Now delete the control02.ctl
  `CONTROL_FILES=/location1/ctrl01.ctl`
- Save the pfile
  `:wq`
  `[oracle@...dbs]$`

Step3: Go to physical location of controlfiles and remove the control02.ctl

- `[oracle@.....~]$ rm control02.ctl`
- `[oracle@.....~]$`
- Restart the database.
  `[oracle@....~]$ sqlplus / as sysdba`

```
Sql> startup
```

Redolog file Demo

1. Theory

- Redolog files are mainly used for recovering a database and also to ensure data commit.
- If a redolog file is lost, it will lead to data loss.
- Oracle recommends to maintain a min of 2 redolog groups with min of 2 members in each group.
- We cannot have different sizes for members in the same group.
- Redo log files have three states. All these states may change in cyclic order.

CURRENT	---	Currently Getting Filled.
ACTIVE	---	Filled and Ready For Recovery.
INACTIVE	---	Empty which is Not Required For Recovery.
UNUSED	---	New Redolog Got Created.

Note: in current state we cannot do any changes like (rename, relocate and drop)

The dictionary view used for redolog are: V$log and v$logfile.

SYNTAX:

For finding current group with size.

- desc v$log
- Select group#, bytes/1024/1024 from v$log;

For finding redolog status.

- Select group# ,status from v$log;

For finding out log files with group.

- desc v$logfile;
    - select group#, member from v$logfile;

2. Creating Redolog Group:

First identify the location of redo logsfiles.

- select group#, member from v$logfile;

For example:

```
SQL> select group#,member from v$logfile;
```

To create redolog group:

Example(1):

```
SQL> alter database add logfile group 4('/u01/db/oraprod/oradata/prod/redo04.log') size 50m;

SQL> Database altered.
```

- After check whether log file group is created or not.
- Select group#,member from v$logfile;

```
SQL> select group#,member from v$logfile;
```

Note: Here you can see the redolog group 4 is created.

3. Adding New Redolog File to Existing Group.

In previous example (1) we have created redolog group 4.

Now here we will add member to the redolog group 4.

Example(2):

```
SQL> alter database add logfile member '/u01/db/oraprod/oradata/prod/redo04a.log' to group 4;

SQL> Database altered.
```

Now check whether log file member is added to the redolog group 4.

```
SQL> select group#,member from v$logfile;
```

Note: Here we can observe the logfile member added to group 4.

4. Dropping the redolog member from redolog group:

Note: While dropping member from a group we have to drop it logically, and remove physically.

- In example (2) we had added log file member to group 4.
- Now in this example we will drop the log file member which we have created.

Example(3):

```
SQL> alter database drop log file member '/u01/db/oraprod/oradata/prod/redo04a.log';

SQL> Database altered.
```

- Now check whether log file has been dropped or not.

```
SQL> select group#,member from v$logfile
```

NOTE: As mentioned in above not now we have to remove the dropped log file member physically.

- First exit from the sql
- Then go to the location of redo log files.

```
[oraprod@prod prod]$ cd /u01/db/oraprod/oradata/prod/
```

- There you will find the dropped log member, present physically.

```
[oraprod@prod prod]$  ls

appstek01.dbf   archive      control03.ctl   nohup.out   redo02.log   redo04a.log
sysaux01.dbf   temp01.dbf    users01.dbf

appstek02.dbf   control01.ctl   example01.dbf  redo01.log   redo03.log   redo04.log
system01.dbf  undotbs01.dbf
```

- Remove the redo log group member which we had droped.

```
[oraprod@prod prod]$ rm -rf redo04a.log.
```

- After removing the log member check again whether it is deleted or not.

5. Reusing the Dropped Member in a Group

- In example (3) we had dropped the log file member in a group.
- In this example we will see how to reuse the dropped log file member.

Example(4):

SQL> alter database add logfile member '/u01/db/oraprod/oradata/prod/redo04a.log' reuse to group 4;

SQL> database altered.

- Now check whether the log file member reused or not.

SQL> select group#,member from v$logfile;

6. Renaming or relocate a redolog member

Note: While renaming or relocating a redo log member we have change it logically and physically.

- We have to shut down the database.
- First we can change it physically.
- Exit from sql.
- From os level go to the location of redolog files.

```
[oraprod@prod ~]$ cd /u01/db/oraprod/oradata/prod
```

- Move the logfile member which you have to rename.

```
[oraprod@prod prod]$ mv redo04b.log redo04a.log
```

- Now check whether the file is moved or not.

- Now rename it logically.
- Now connect to sql.
- Startup mount.

```
SQL> alter database rename file '/u01/db/oraprod/oradata/prod/redo04a.log' to

'/u01/db/oraprod/oradata/prod/redo04b.log';
```

- Now check whether the logfile member had renamed or not.

```
SQL> select group#,member from v$logfile;
```

- After that alter the database.

```
Alter database open;
```

7. Dropping redolog group

Note: while dropping a redolog group we have to drop it logically and remove physically.

In example(1),example(2) we had created redolog group, added logfile member to that group.

- If we drop a logfile group the members present in that group also automatically drops.

```
SQL> alter database drop logfile group 4;

SQL> Database altered
```

- now check whether the logfile group had dropped or not.

```
SQL> select group#,member from v$logfile;
```

- Now remove physically.
- Exit from sql, go to the location of redo log files.

```
[oraprod@prod prod]$ cd /u01/db/oraprod/oradata/prod
```

- Now remove the group and its member present in redo log location.

```
[oraprod@prod prod]$ rm redo04.log redo04b.log
```

- Now check whether the redo log files removed or not.

8. Resizing redo log member

Step1: First see the size of the current logfiles:

```
SQL> select group#, bytes, status from v$log;
GROUP#        BYTES   STATUS
———- ———- ——————-
1             1048576        INACTIVE
2             1048576        CURRENT
3             1048576         INACTIVE
```

Logs are 1MB from above, let’s size them to 10MB.

Step2: Retrieve all the log member names for the groups:

```
SQL> select group#, member from v$logfile;
GROUP#        MEMBER
————— ———————————————-
1          /usr/oracle/dbs/log1PROD.log
2         /usr/oracle/dbs/log2PROD.log
3         /usr/oracle/dbs/log3PROD.log
```

Step3: Let’s create 3 new log groups and name them groups 4, 5, and 6, each 10MB in size:

```
Ex: sql> ALTER DATABASE ADD LOGFILE GROUP 4(‘/usr/oracle/dbs/log4prod.log’) size 10m;
        Database altered
        Sql>ALTER DATABASE ADD LOGFILE GROUP 5(‘/usr/oracle/dbs/log5prod.log’)
        size 10m;
        Database altered

        Sql> ALTER DATABASE ADD LOGFILE GROUP 6(‘/usr/oracle/dbs/log6prod.log’) size 10m;
        Database altered
```

Step4: Now drop the old log groups(1,2 & 3).

The members of the groups should be in inactive state to drop.

```
Sql> select group#,status from v$log;
        Group# Status
```

1	inactive
2	active
3	inactive
4	current
5	inactive
6	inactive

Note: Here each group has single member only.

The status of members is changed when logfile is switched.

9. Manual switching of logfile:

```
Sql> alter system switch logfile;
System altered
Sql>
```

10. Views:

- V$CONTROLFILE
- V$LOGFILE
- V$LOG