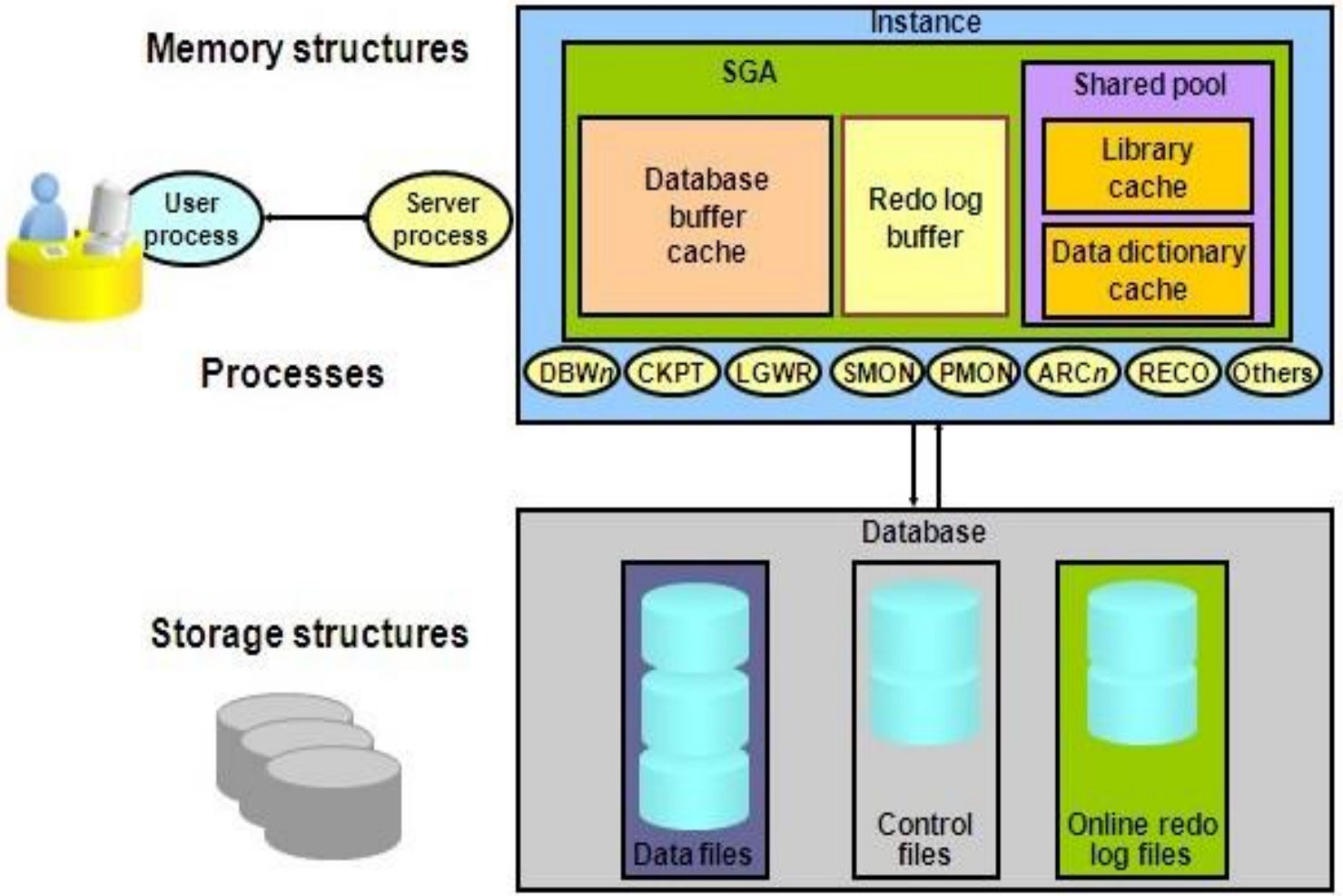
SAROJ ARYAL

MSC. IT Madrass University

OCP 10g, RHCE 6.0

Database Administration(DBA)
CSIT 7 Semister

ORACLE DATABASE SERVER



Oracle Instance and Database

- A database instance is a set of memory structures and background processes that manage database files. The instance manages its associated data and serves the users of the database.
- Every running Oracle database is associated with at least one Oracle database instance. Because an instance exists in memory and a database exists on disk, an instance can exist without a database and a database can exist without an instance

Database Instance Structure

- When an instance is started, Oracle Database allocates a memory area called the system global area(SGA) and starts one or more background processes. The SGA serves various purposes, including the following:
- Maintaining internal data structures that are accessed by many processes and threads concurrently
- Caching data blocks read from disk
- Buffering redo data before writing it to the online redo log files
- Storing SQL execution plans

Oracle Client

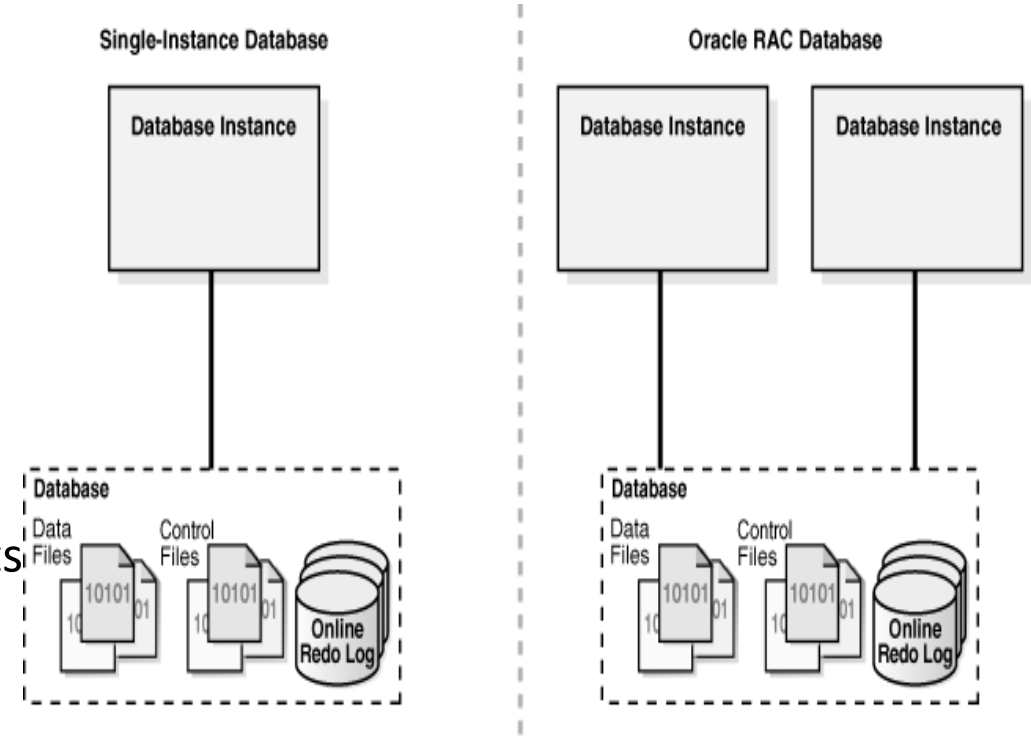
- The client is a database application that initiates a request for an operation to be performed on the database server. It requests, processes, and presents data managed by the server. The client workstation can be optimized for its job. For example, it might not need large disk capacity, or it might benefit from graphic capabilities.

Database Instance Configurations

- You can run Oracle Database in either of the following mutually exclusive configurations:
- Single-instance configuration

  A one-to-one relationship exists between the database and an instance.
- Oracle Real Application Clusters (Oracle RAC) configuration

  A one-to-many relationship exists between the database and instances



Oracle Database and DBMS

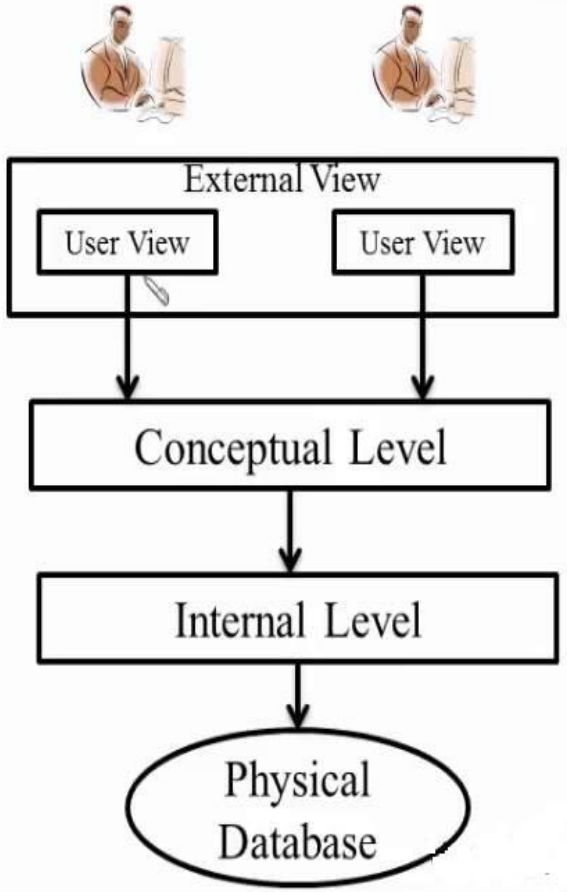
- An Oracle **database** is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. It is a set of physical files on disk created by the CREATE DATABASE statement.
- A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data.

Advantages of DBMS

- Data abstraction and independence
- Data security
- A locking mechanism for concurrent access
- An efficient handler to balance the needs of multiple applications using the same data
- The ability to swiftly recover from crashes and errors, including restartability and recoverability
- Robust data integrity capabilities
- Logging and auditing of activity
- Simple access using a standard application programming interface (API)
- Uniform administration procedures for data

Architecture of DBMS

- Logical DBMS Architecture or Three Level Architecture of DBMS
  - The External or View Level
  - The conceptual or Global Level
  - The Internal or Physical Level
- Physical DBMS Architecture



- External Level or View Level

  It is used for end user. User can interact with view level. This level describes that part of db that is relevant to each user.

Conceptual level or Logical Level

This describes what data is stored in db and relationship among the data.

It describes:

a) all entities, attributes and their relationship

b) constrains on the data

c) security and integrity information

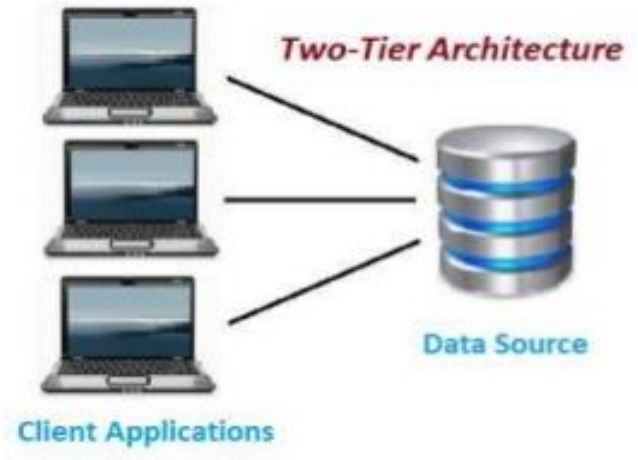
- Internal or Physical Level or Storage Level

  It is the physical representation of DB. This level describes how the data is stored in DB . It covers the data structures and file organization.

- 1, 2-tier, 3-tier and n-tier DBMS architecture

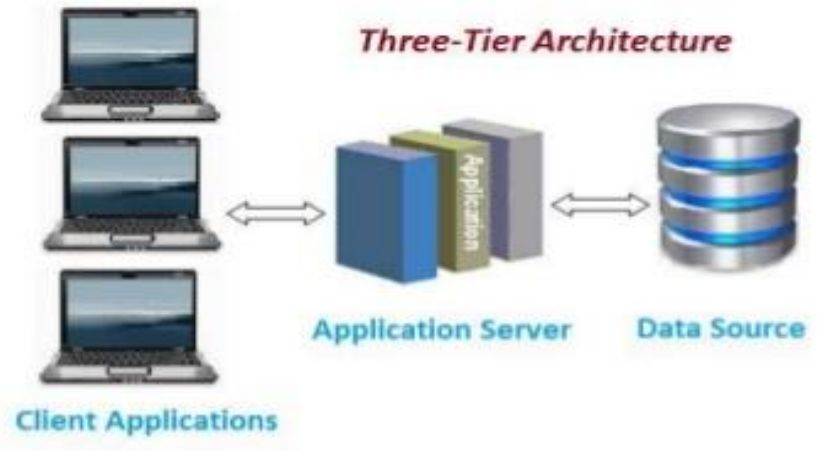
2-TIER ARCHITECTURE

- It is client-server architecture
- Direct communication
- Run faster(tight coupled)



3-TIER ARCHITECTURE

- **Web based application**
- **Three layers:**
  - **1) Client layer**
  - **2) Business layer**
  - **3) Data layer**



Data Independence

Data independency also known as data abstraction is one of the strong feature of three layered architecture of DBMS. It is the ability to change the schema at one level of the database system without affecting the other levels.

Types:

- Logical data Independency
- Physical data independency

Logical data independency is the ability to change the conceptual layer without affecting the external layer.

Eg: insert/delete/update data

- Physical data Independency is the ability to change the internal layer without affecting the conceptual or external layer.

Eg: change of storage location, addition of indexes, tablespace changes etc.

DBA Roles and Responsibilities

- A database administrator's (DBA) primary job is to ensure that data is available, protected from loss and corruption, and easily accessible as needed.

Below are some of the chief responsibilities that make up the day-to-day work of a DBA:

1) Software installation and Maintenance

This includes installation of oracle software and database, its maintenance, application of patches, etc.

2) **Data Extraction, Transformation, and Loading**

Known as ETL and it refers to efficiently importing large volumes of data that have been extracted from multiple systems into a data warehouse environment.

3) **Database Backup and Recovery**

DBAs create backup and recovery plans and procedures based on industry best practices.

In the case of a server failure or other form of data loss, the DBA will use existing backups to restore lost information to the system. Different types of failures may require different recovery strategies, and the DBA must be prepared for any eventuality.

4) **Security**

A DBA needs to know potential weaknesses of the database software and the company's overall system and work to minimise risks.

In the case of a security breach or irregularity, the DBA can consult audit logs to see who has done what to the data

5) **Authentication**

Setting up employee access and control who has access and what type of access they are allowed.

6) **Capacity Planning**

The DBA needs to know how large the database currently is and how fast it is growing in order to make predictions about future needs.

7) **Performance Monitoring**

Monitoring databases for performance issues is part of the on-going system maintenance a DBA performs. If some part of the system is slowing down processing, the DBA may need to make configuration changes to the software or add additional hardware capacity.

8) **Database Tuning**

Performance monitoring shows where the database should be tweaked to operate as efficiently as possible.

9) **Troubleshooting**

DBAs are on call for troubleshooting in case of any problems. Whether they need to quickly restore lost data or correct an issue to minimise damage, a DBA needs to quickly understand and respond to problems when they occur.

SQL*Plus Overview

- SQL*Plus stands for Structured Query Language. It is command-line interface to the Oracle database which allows us to enter and execute SQL statements and PL/SQL code blocks.

Advantages:

With SQL*Plus, we can do the following:

- Issue a SELECT query and view the results
- Insert, update, and delete data from database tables
- Submit PL/SQL blocks to the Oracle server for execution
- Issue DDL commands, such as those used to create, alter, or drop database
- objects such as tables, indexes, and users
- Execute SQL*Plus script files
- Write output to a file
- Execute procedures and functions that are stored in a database

Operators

- An operator manipulates individual data items and returns a result. The data items are called **operands** or **arguments**. Operators are represented by special characters or by keywords. For example, the multiplication operator is represented by an asterisk (*) and the operator that tests for nulls is represented by the keywords IS NULL.

- **Unary and Binary Operators**

The two general classes of operators are:

- unary

  A unary operator operates on only one operand. Eg:

  SELECT * FROM orders WHERE qtysold = -1; SELECT * FROM emp WHERE sal < 0;

- A binary operator operates on two operands.

  SELECT sal + comm FROM emp WHERE SYSDATE - hiredate > 365;

  UPDATE emp SET sal = sal * 1.1;

Arithmetic Operators

+, - , /,*

Concatenation Operator

SELECT 'Name is ' || ename FROM emp;

Comparison Operators

=

!= or ^= or < >

>

<

>=

<=

IN	Equivalent to "= ANY
NOT IN	Equivalent to “!=ALL”
ANY	Compares a value to each value in a list or returned by a query.

Eg: SELECT * FROM emp WHERE sal = ANY (SELECT sal FROM emp WHERE deptno = 30);

BETWEEN

[Not] greater than or equal to *x* and less than or equal to *y*.

```
SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000;
```

EXISTS

TRUE if a subquery returns at least one row.

```
SELECT ename, deptno FROM dept WHERE EXISTS (SELECT * FROM emp WHERE dept.deptno =
  emp.deptno);
```

IS [NOT] NULL

Tests for nulls. This is the only operator that you should use to test for nulls.

```
SELECT ename, deptno FROM emp WHERE comm IS NULL;
```

ALL

Compares a value to every value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, >=.

```
SELECT * FROM emp WHERE sal >= ALL ( 1400, 3000);
```

LIKE

Used for matching the content

```
SELECT * FROM tab1 WHERE NAME LIKE ‘BIKASH%‘
```

It uses ESCAPE keyword to filter special character

To search for employees with the pattern 'A_B' in their name:

```
SELECT ename FROM emp WHERE ename LIKE '%A\_B%' ESCAPE '\';
```

Logical Operators: NOT, AND, OR

- A logical operator combines the results of two component conditions to produce a single result based on them or to invert the result of a single condition.

NOT

Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.

```
SELECT * FROM emp WHERE NOT (job IS NULL);
SELECT * FROM emp WHERE NOT (sal BETWEEN 1000 AND 2000);
```

AND

Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE. Otherwise returns UNKNOWN.

```
SELECT * FROM emp WHERE job = 'CLERK' AND deptno = 10;
```

OR

Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE. Otherwise returns UNKNOWN.

```
SELECT * FROM emp WHERE job = 'CLERK' OR deptno = 10;
```

Producing Readable Output with SQL*Plus

Produce queries that require an input variable

Use SQL*Plus substitution variables to temporarily store values.

- Single ampersand (&)
- Double ampersand (&&)
- DEFINE and ACCEPT commands

- Pass variable values between SOL statements.

  & is used to create a temporary substitution variable that will prompt you for a value every time it is reffered.

  Eg: SQL> SELECT sal FROM emp WHERE ename = '&NAME';

  Enter value for name: SCOTT
  old 1: SELECT sal FROM emp WHERE ename = '&NAME'
  new 1: SELECT sal FROM emp WHERE ename = 'SCOTT'

  SAL

  3000

- && - is used to create a permanent substitution variable for the session. Once you have entered a value its value will used every time the variable is referenced.

  Eg: SQL> SELECT sal FROM emp WHERE ename LIKE '&&NAME';

  Enter value for name: SCOTT
  old 1: SELECT sal FROM emp WHERE ename LIKE '&&NAME'
  new 1: SELECT sal FROM emp WHERE ename LIKE 'SCOTT'

  SAL

  3000

  SQL> /

  old 1: SELECT sal FROM emp WHERE ename LIKE '&&NAME'
  new 1: SELECT sal FROM emp WHERE ename LIKE 'SCOTT'

  SAL

  3000

- Specifying Column Names, Expressions, and Text at Runtime

  Use substitution variables to supplement the following:
  - WHERE condition
  - ORDER BY clause
  - Column expression
  - Table name
  - Entire SELECT statement

  SELECT empno, &column_name FROM emp WHERE &condition;

  Enter value for column_name: job

  Enter value for condition: deptno=10

  old 3: WHERE &condition

  new 3: WHERE deptno=10

```
SELECT empno, ename, &column_name
FROM emp WHERE &condition ORDER BY
&order_column ;
```

```
Enter value for column_name : sal
Enter value for condition : sal >= 3000
Enter value for order aolumn : ename
```

The ACCEPT Command

Creates a customized prompt when accepting user input. When using the ACCEPT command, use single quotation marks (' ') to enclose a string that contains an embedded space.

ACCEPT dept PROMPT ‘ Provide the department name: ‘

SELECT * FROM dept WHERE dname = UPPER(‘&dept’) ;

Provide the department name: sales

old 3: WHERE dname= UPPER('&dept')

new 3: WHERE dname= UPPER('sales')

- **Using the DEFINE Command**

  Use the DEFINE cornmand to create a variable and then use the variables:

  DEFINE occupation=clerk

  SELECT * FROM emp WHERE job = UPPER('&occupation');

  old 3: WHERE job = UPPER('&occupation')

  new 3: WHERE job = UPPER('clerk')
- **Using UNDEFINE Commands**

  A variable remains defined until you either:

  -- Use the UNDEFINE command to clear it

  -- Exit SQL*Plus

We can verify the changes with the DEFINE command.

Data types in Oracle

- **CHAR Datatype**

  The CHAR datatype stores fixed-length character strings.

- **VARCHAR2 and VARCHAR Datatypes**

  The VARCHAR2 datatype stores variable-length character strings. When you create a table with a VARCHAR2 column, you specify a maximum string length (in bytes or characters) between 1 and 4000 bytes for the VARCHAR2 column.

  The VARCHAR datatype is synonymous with the VARCHAR2 datatype. To avoid possible changes in behavior, always use the VARCHAR2 datatype to store variable-length character strings.

- LOB Character Datatypes

  The LOB datatypes stores character data and they can store up to 8 terabytes of character data.

  The LOB datatypes BLOB, CLOB, NCLOB, and BFILE enable you to store and manipulate large blocks of unstructured data (such as text, graphic images, video clips, and sound waveforms) in binary or character format. They provide efficient, random, piece-wise access to the data.
- NUMBER Datatype

  The number datatypes store positive and negative fixed and floating-point numbers, zero, infinity
- DATE Datatype

  The DATE datatype stores point-in-time values (dates and times) in a table. The DATE datatype stores the year (including the century), the month, the day, the hours, the minutes, and the seconds (after midnight), format is DD-MON-YY

- **Files in Oracle Database**

  Parameter Files

  Alert Log Files

  Trace Files

  Data Files

  Temp Files

  Control Files

  Redo Log Files

  Password Files

Control Files

- A control file is a binary file which contains information about the associated database that is required for access by an instance, both at startup and during normal operation. Control file information can be modified only by Oracle; no database administrator or user can edit a control file.

Among other things, a control file contains information such as:

- The database name
- The timestamp of database creation
- The names and locations of associated datafiles and redo log files
- Tablespace information
- The log history
- Archived log information
- Backup set and backup piece information
- Backup datafile and redo log information
- Datafile copy information
- The current log sequence number
- Checkpoint information

Control files also record information about checkpoints. Every three seconds, the checkpoint process (CKPT) records information in the control file about the checkpoint position in the redo log. This information is used during database recovery to tell Oracle that all redo entries recorded before this point in the redo log group are not necessary for database recovery; they were already written to the datafiles.

Multiplexed Control Files

- Oracle enables multiple, identical control files to be open concurrently and written for the same database. By storing multiple control files for a single database on different disks, we can safeguard against a single point of failure with respect to control files. If a single disk that contained a control file crashes, then the current instance fails when Oracle attempts to access the damaged control file. However, when other copies of the current control file are available on different disks, an instance can be restarted without the need for database recovery.
- We can have maximum of 5 copies of control files in one database.

It is strongly recommended that you adhere to the following:

- Use multiplexed control files with each database
- Store each copy on a different physical disk
- Use operating system mirroring
- Monitor backups

Multiplexing Control Files Using init.ora

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

Eg: add following lines in init.ora file

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

- Start up the database using init.ora file
- For permanent defining control file in multiple locations, create spfile from init.ora file

- **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:

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.
- Shut down the database.

  SQL> SHUTDOWN NORMAL
- Copy an existing control file to the new location:

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

  /ora04/oradata/MYDB/ctrlMYDB04.ctl
- Start the instance.

  SQL> STARTUP

Using OMF to Manage Control Files

- DB_CREATE_ONLINE_LOG_DEST_n is specified in init.ora file
- *n* is the number of desired control files to be created. The actual names of the control files are system generated and can be found in the alert logs located in $ORACLE_HOME/admin/bdump.

Querying Control File Information

- V$CONTROLFILE lists the names of the control files for the database
- You can also use the SHOW PARAMETER command to retrieve the names of the control files.

SQL> show parameter control_files

Manually Creating Control File

- ALTER DATABASE **BACKUP CONTROLFILE**TO '/oracle/**backup/control**.bkp';

OR

ALTER DATABASE **BACKUP CONTROLFILE** TO TRACE;

Redo Log Files

- Redo LOg files:

  Is the transaction log of the database. By following the redo log file we can take database to any desired state.
- 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. For example, if you change a salary value in an employee table, you generate a redo record containing change vectors that describe changes to the database.

- Redo records are buffered in a circular fashion in the redo log buffer of the SGA and are written to one of the redo log files by the Log Writer (LGWR) database background process. Whenever a transaction is committed, 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) to identify the redo records for each committed transaction.

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
- Manual log switch:

ALTER SYSTEM SWITCH LOGFILE

Multiplexing Redo Log Files

- To protect against a failure involving the redo log itself, Oracle Database allows a **multiplexed** redo log, meaning that two or more identical copies of the redo log can be automatically maintained in separate locations. For the most benefit, these locations should be on separate disks.
- When redo log files are multiplexed, LGWR concurrently writes the same redo log information to multiple identical redo log files, thereby eliminating a single point of redo log failure.
- Syntax:

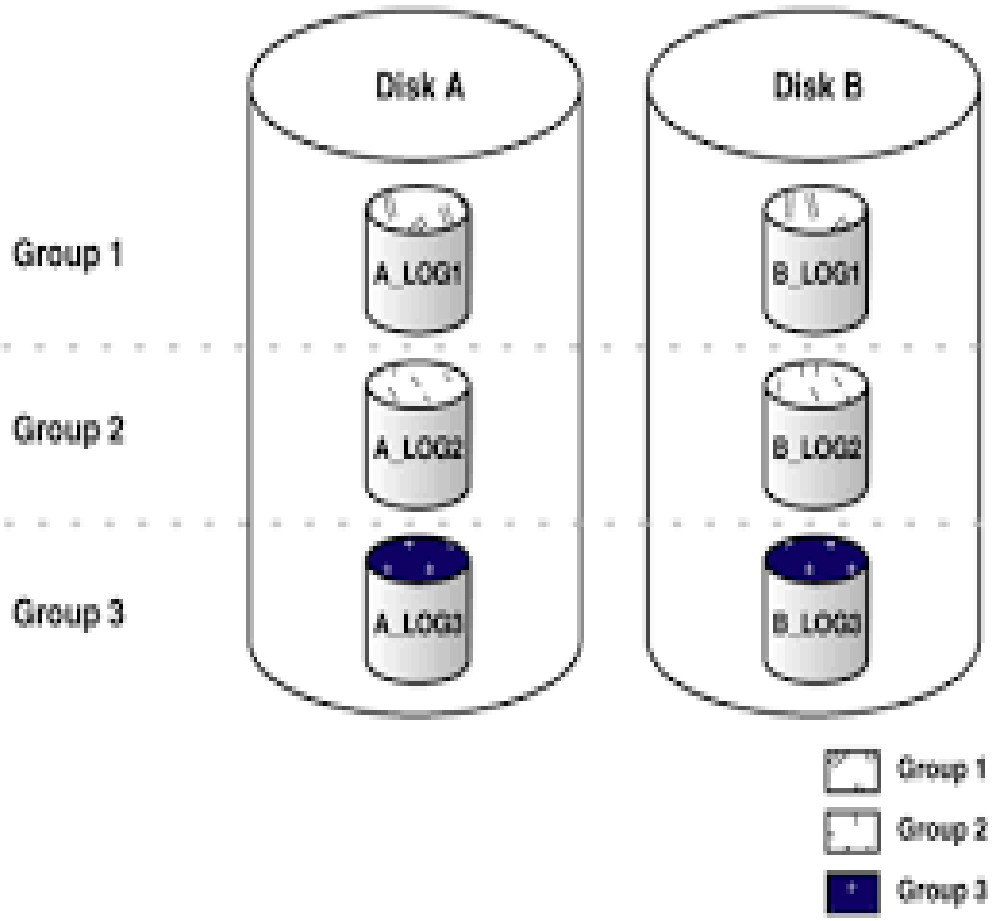
```
ALTER DATABASE ADD LOGFILE GROUP 1
('/oracle1/dbs/log101.log', '/oracle2/dbs/log102.log') SIZE
4M;

ALTER DATABASE ADD LOGFILE GROUP 2
('/oracle1/dbs/log201.log', '/oracle2/dbs/log202.log') SIZE
4M;
```

Multiplexing is implemented by creating *groups* of redo log files. A **group** consists of a redo log file and its multiplexed copies. Each identical copy is said to be a **member** of the group. Each redo log group is defined by a number, such as group 1, group 2, and so on.

Here, A_LOG1 and B_LOG1 are both members of Group 1, A_LOG2 and B_LOG2 are both members of Group 2, and so forth. Each member in a group must be exactly the same size.

First LGWR writes concurrently to both A_LOG1 and B_LOG1. Then it writes concurrently to both A_LOG2 and B_LOG2, and so on. LGWR never writes concurrently to members of different groups



Creating New Groups

- You can create and add more redo log groups to the database by using the ALTER DATABASE command:
- ALTER DATABASE ADD LOGFILE GROUP 3 (‘/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;

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:

ALTER DATABASE ADD LOGFILE MEMBER

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

Renaming Log Members

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

Dropping Redo Log Groups

- ALTER DATABASE DROP LOGFILE GROUP 3;

  Note: Group must have log file in INACTIVE state. If its in active or current state/status, run the following command till the desired log group's log file is INACTIVE:

ALTER SYSTEM SWITCH LOGFILE;

Dropping Redo Log 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';

Log status

- UNUSED - Online redo log has never been written to. This is the state of a redo log that was just added, or just after a RESETLOGS, when it is not the current redo log.
- CURRENT - Current redo log. This implies that the redo log is active. The redo log could be open or closed.
- ACTIVE - Log is active but is not the current log. It is needed for crash recovery. It may be in use for block recovery. It may or may not be archived.
- CLEARING - Log is being re-created as an empty log after an ALTER DATABASE CLEAR LOGFILE statement. After the log is cleared, the status changes to UNUSED.
- CLEARING_CURRENT - Current log is being cleared of a closed thread. The log can stay in this status if there is some failure in the switch such as an I/O error writing the new log header.
- INACTIVE - Log is no longer needed for instance recovery. It may be in use for media recovery. It might or might not be archived.

Archive Log Files

- When an online redo log file is full, and LGWR starts writing to the next redo log file, ARC*n* copies the completed redo log file to the archive destination and it is called Archiving.
- It is possible to specify more than one archive destination. The LGWR process waits for the ARC*n* process to complete the copy operation before overwriting any online redo log file.

Check the archival or no archive status of database:

SQL> archive log list;

Setting ARCHIVELOG