Unit -1 Introduction to Oracle RDBMS

Overview of Oracle Database Architecture

Oracle Database is an object-relational database management system developed and marketed by Oracle Corporation. Oracle Database is commonly referred to as Oracle RDBMS or simply Oracle.

An **Oracle Database** consists of a **database** and at least one **instance**. An instance, or database instance, is the combination of **memory** and **processes** that are a part of a running installation and a database is a **set of files** that store data.

The following picture illustrates the Oracle Database server architecture.

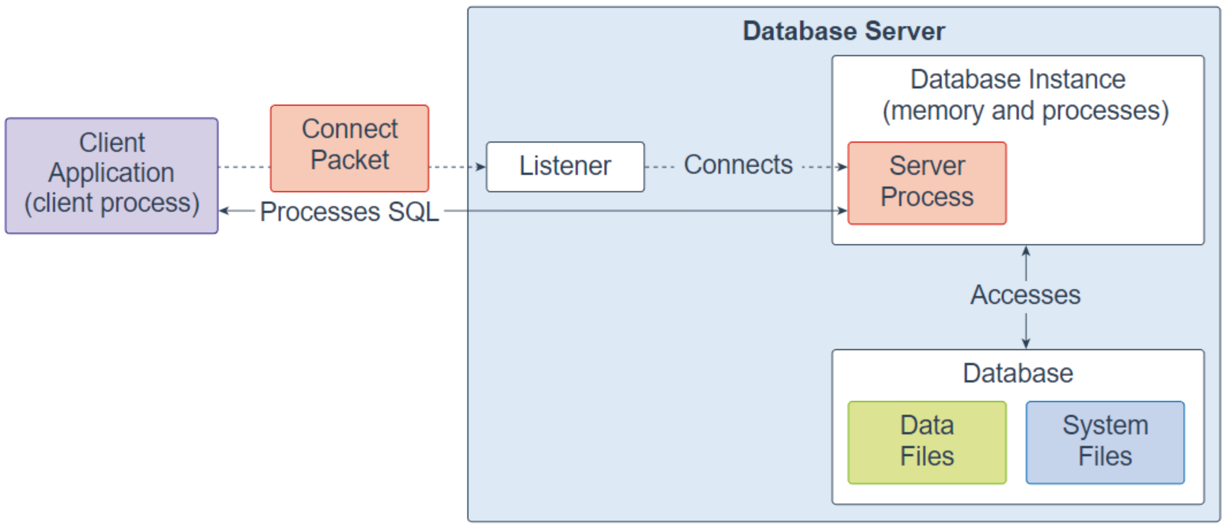


Fig: Oracle Server Architecture

Sometimes, a database instance is referred to as an entire running database. However, it is important to understand the distinctions between the two.

- **First, you can start a database instance without having it accessing any database files**. This is how you create a database, starting an instance first and creating the database from within the instance.
- **Second, an instance can access only one database at a time**. When you start an instance, the next step is to mount that instance to a database. And an instance can mount only one database at a single point in time.
- **Third, multiple database instances can access the same database**. In a clustering environment, many instances on several servers can access a central database to enable high availability and scalability.
- **Finally, a database can exist without an instance**. However, it would be unusable because it is just a set of files.

Physical storage structures

The physical storage structures are simply files that store data.

When you execute a CREATE DATABASE statement to create a new database, Oracle creates the following files:

- **Data files:** data files contain real data, e.g., sales order and customer data. The data of logical database structures such as tables and indexes are physically stored in the data files.
- **Control files:** every database has a control file that contains metadata. The metadata describes the physical structure of the database including the database name and the locations of data files.
- **Online redo log files:** every database has an online redo log that consists of two or more online redo log files. An online redo log is made up of redo entries that record all changes made to the data.

Besides these files, an Oracle database includes other important files such as **parameter files, network files, backup files, and archived redo log files for backup and recovery.**

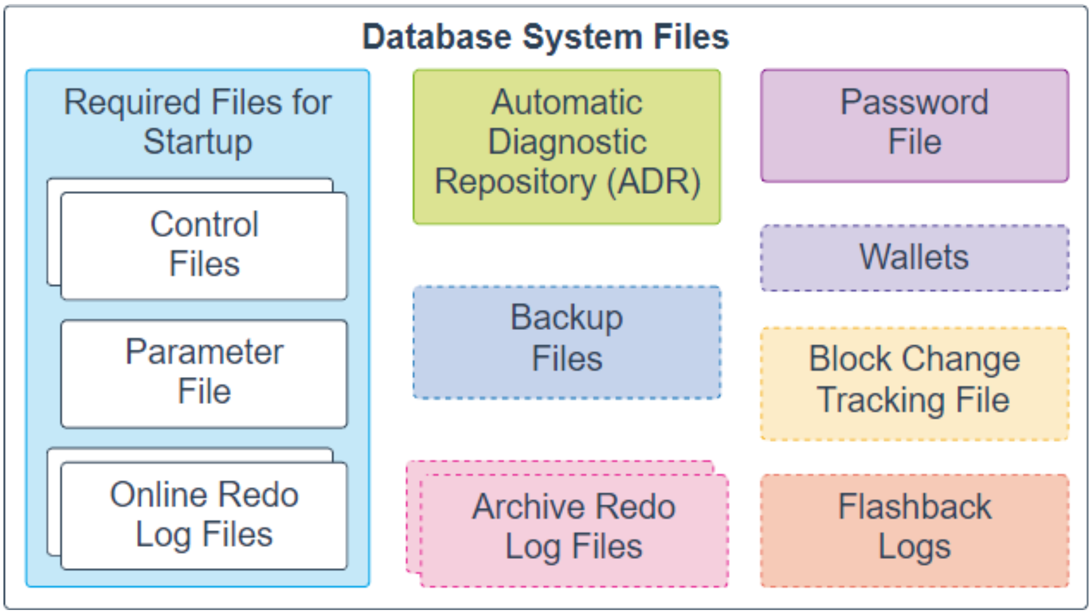


Fig: Oracle Database Physical Structure

Logical Storage Structures

Oracle Database uses a logical storage structure for fine-grained control of disk space usage.

The following are logical storage structures in an Oracle Database:

- **Data blocks**: a data block corresponds to a number of bytes on the disk. Oracle stores data in data blocks. Data blocks are also referred to as logical blocks, Oracle blocks or pages.
- **Extents**: An extent is a specific number of logically contiguous data blocks used to store the particular type of information.
- **Segments:** a segment is a set of extents allocated for storing database objects, e.g., a table or an index.

- T**ablespace**s: a database is divided into logical storage units called tablespaces. A tablespace is a logical container for a segment. Each tablespace consists of at least one data file.

The following picture illustrates segments, extents and data blocks within a tablespace:

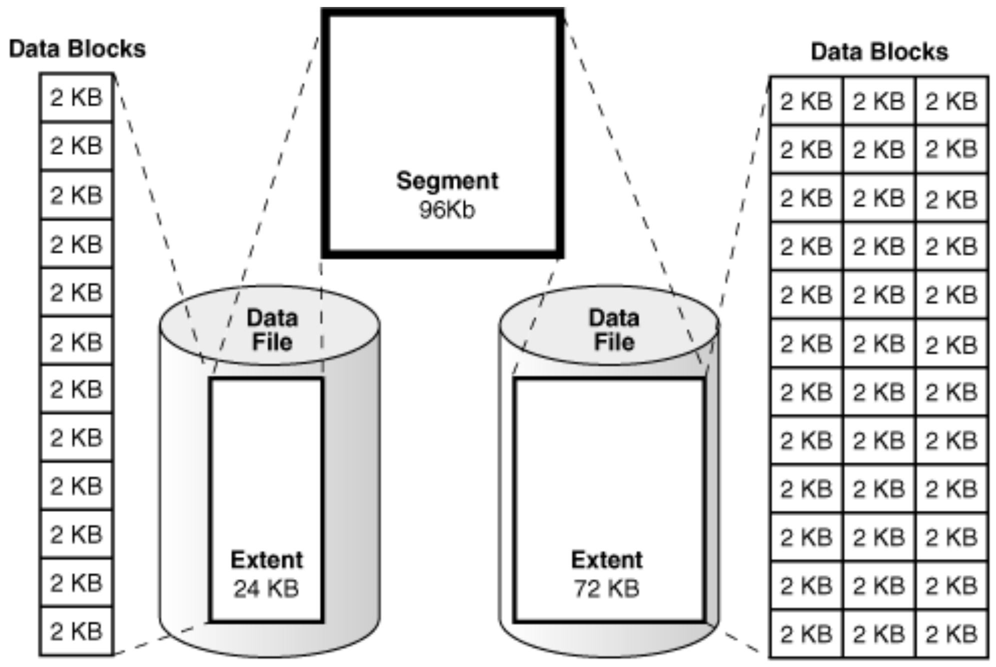


Fig: Oracle Database Logical Structure

And the next figure shows the relationship between logical and physical storage structures:

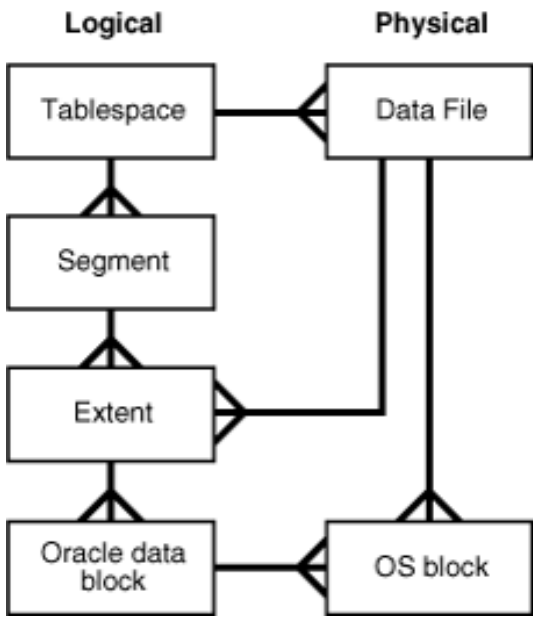
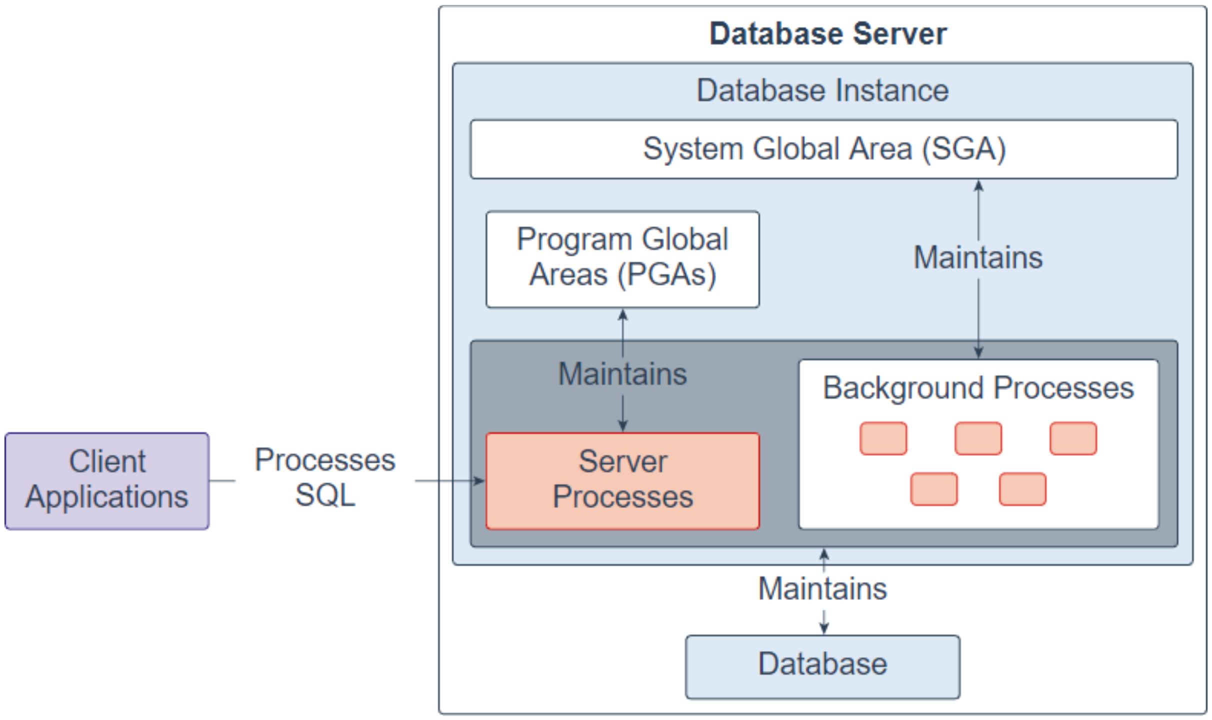


Fig: Relationship between physical and logical structure

<u>Database Instance</u>

A Database Instance is an interface between client applications (users) and the database. An Oracle instance consists of **three main parts: System Global Area (SGA), Program Global Area (PGA), and background processes.**



The SGA is a shared memory structure allocated when the instance started up and released when it is shut down. The SGA is a group of shared memory structures that contain data and control information for one database instance.

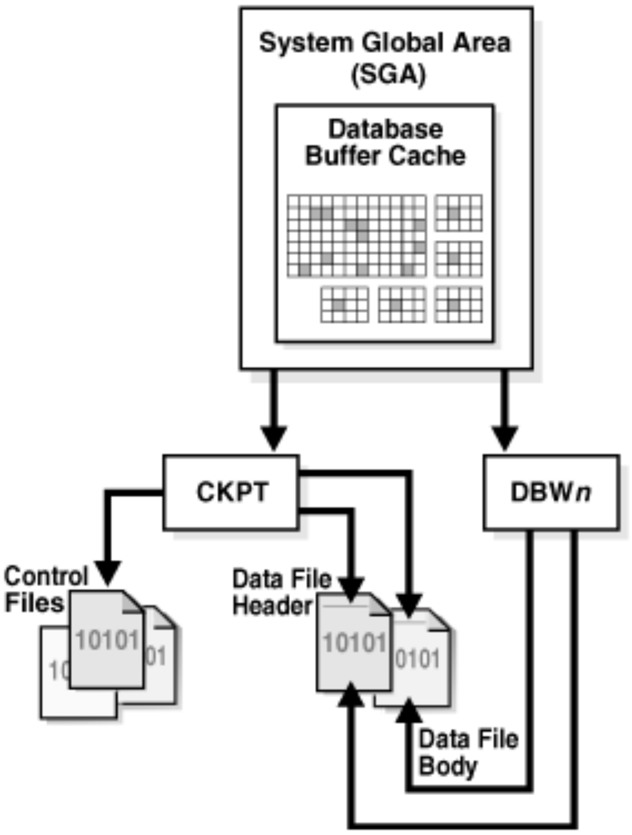
Different from the SGA, which is available to all processes, **PGA is a private memory area allocated to each session when the session started and released when the session ends.**

<u>Major Oracle Database's background processes</u>

The following are the major background processes of an Oracle instance:

- **PMON is the process monitor that regulates all other processes.** PMON cleans up abnormally connected database connections and automatically registers a database instance with the listener process. PMON is a process that must be alive in an Oracle database.
- **SMON is the system monitor process that performs system-level clean-up operation.** It has two primary responsibilities including automatically instance recovery in the event of a failed instance, e.g., power failure and cleaning up of temporary files.
- **DBWn is the database writer. Oracle performs every operation in memory instead of the disk because processing in memory is faster and more efficient than on disk.** The DBWn process reads data from disk and writes it back to the disk. An Oracle instance has many database writers DBW0, DBW1, DBW2, and so on.
- **CKPT is the checkpoint process.** In Oracle, data that is on disk is called block and the data which in memory is called buffer. When a block is written to the buffer and

changed, the buffer becomes dirty, and it needs to be written down to the disk. The CKPT process updates the control and data file headers with checkpoint information and signals writing of dirty buffers to disk. Note that Oracle 12c allows both full and incremental checkpoints.



- **LGWR is the log writer process which is the key to the recoverability architecture**. Every change occurs in the database is written out to a file called redo log for recovery purposes. And these changes are written and logged by LGWR process. The LGWR process first writes the changes to memory and then disk as redo logs which then can be used for recovery.
- **ARCn is the archiver process that copies the content of redo logs to archive redo log files**. The archiver process can have multiple processes such as ARC0, ARC1, and ARC3, which allow the archiver to write to various destinations such as D: drive, E drive or other storage.
- **MMON is the manageability monitor process that gathers performance metrics.**
- **MMAN is the memory manager that automatically manages memory in an Oracle database.**
- **LREG is the listener registration process that registers information on the database instance and dispatcher processes with the Oracle Net Listener.**

SQL * Plus

- **SQL * Plus is a tool that allows you to interact directly with the Oracle database.**
- SQL * Plus is an Oracle command line utility that allows users to interactively execute SQL and PL/SQL commands.
- SQL * Plus enables you to manipulate blocks of SQL and PL / SQL formatting the query results, to copy data between tables and to execute SQL and PL / SQL.

Operations you can perform in SQL * Plus

- edit, save, load and execute SQL commands and PL/SQL blocks;
- list tables structures;
- access and transfer data between databases;
- implementation of functions for database managing: the users administration, namespaces table, operations management of archiving and recovery.
- And many more…

SQL * Plus Commands

DESCRIBE Command

Lists the column definitions for the specified table, view, or synonym or the specifications for the specified function procedure.

Syntax:
DESCRIBE table_name
DESCRIBE view_name
DESCRIBE package_name

Example:
DESCRIBE employee

Lists all columns along with data types and other details.

LIST Command

Lists one or more lines of the buffer.

Term	**Description**
n	Lists line *n*.
n m	Lists lines *n* through *m*.
n *	Lists line *n* through the current line.
n LAST	Lists line *n* through the last line.
*	Lists the current line.
* *n*	Lists the current line through line *n*.
* LAST	Lists the current line through the last line.
LAST	Lists the last line.

Syntax:
LIST n | n m | n* …

Example:

1 SELECT name, dept_id, role_id
2 FROM users
3 WHERE role_id= 'MANAGER'
4* ORDER BY dept_id

SQL> LIST 3

Result: 3* WHERE role_id= 'MANAGER'

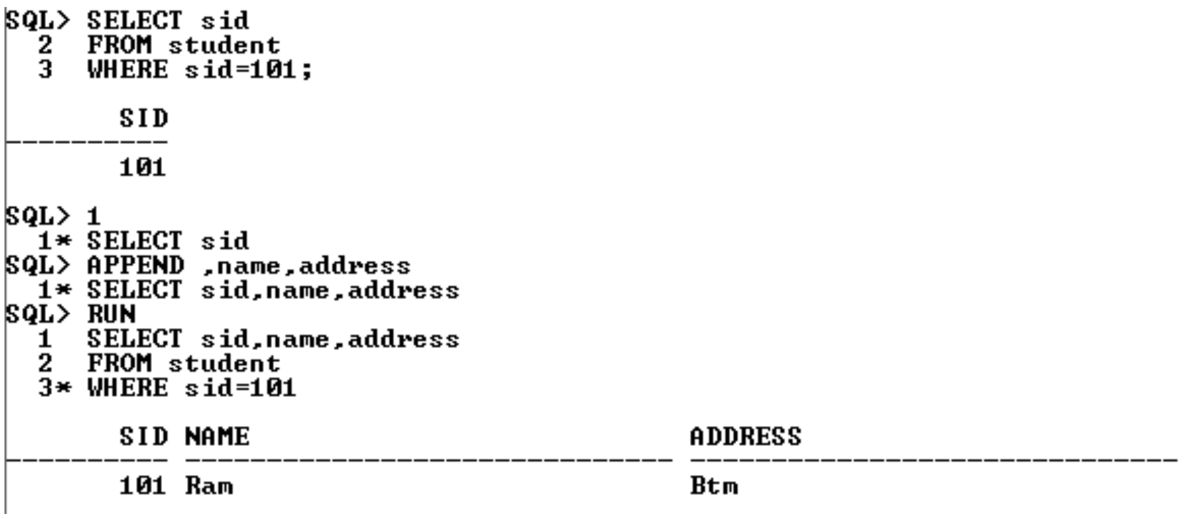
APPEND Command

Adds specified text to the end of the current line in the SQL buffer. To separate text from the preceding characters with a space, enter two spaces between APPEND and text.

To APPEND text that ends with a semicolon, end the command with two semicolons (SQL*Plus interprets a single semicolon as an optional command terminator).

Syntax:
APPEND text
where *text* represents the text to append.

Example



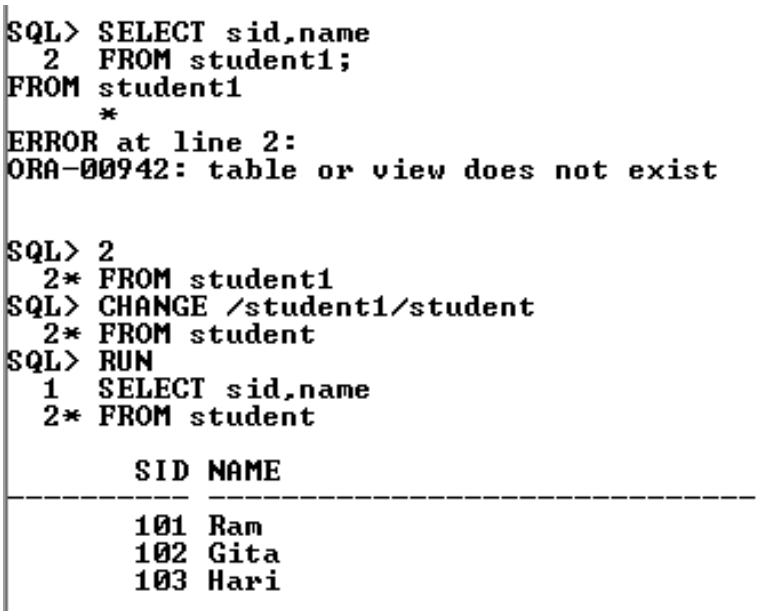
```
SQL> SELECT sid
  2  FROM student
  3  WHERE sid=101;

       SID
----------
       101

SQL> 1
  1* SELECT sid
SQL> APPEND ,name,address
  1* SELECT sid,name,address
SQL> RUN
  1  SELECT sid,name,address
  2  FROM student
  3* WHERE sid=101

       SID NAME                           ADDRESS
---------- ------------------------------ ------------------------------
       101 Ram                            Btm
```

CHANGE Command

Changes text on the current line in the buffer.

Syntax:

```
CHANGE /old/new/
```



```
SQL> SELECT sid,name
  2  FROM student1;
FROM student1
     *
ERROR at line 2:
ORA-00942: table or view does not exist


SQL> 2
  2* FROM student1
SQL> CHANGE /student1/student
  2* FROM student
SQL> RUN
  1  SELECT sid,name
  2* FROM student

       SID NAME
---------- ------------------------------
       101 Ram
       102 Gita
       103 Hari
```

INPUT Command

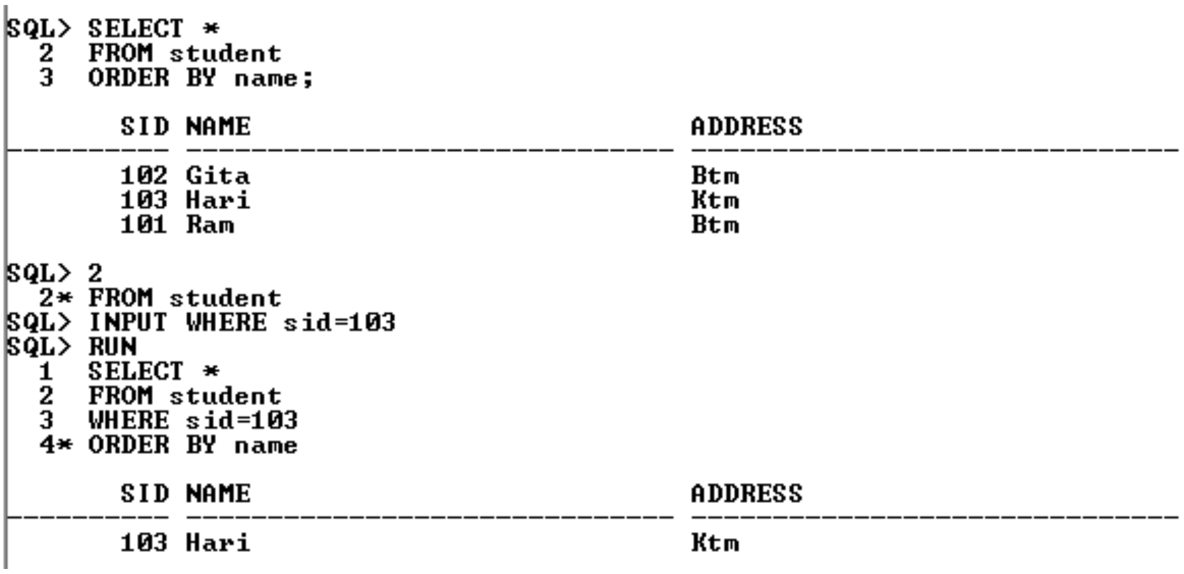
Adds one or more new lines after the current line in the buffer.

To add a single line, enter the text of the line after the command INPUT, separating the text from the command with a space. To begin the line with one or more spaces, enter two or more spaces between INPUT and the first non-blank character of text.

To add several lines, enter INPUT with no text. INPUT prompts you for each line. To leave INPUT, enter a null (empty) line or a period.

Syntax:

INPUT [text]



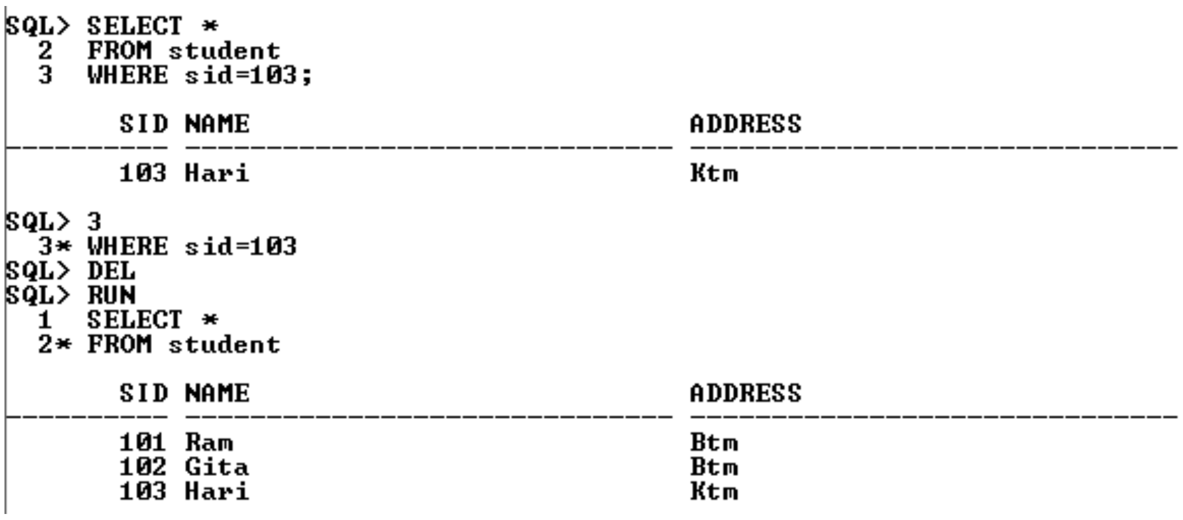
```
SQL> SELECT *
  2  FROM student
  3  ORDER BY name;

       SID NAME                           ADDRESS
---------- ------------------------------ ------------------------------
       102 Gita                           Btm
       103 Hari                           Ktm
       101 Ram                            Btm

SQL> 2
  2* FROM student
SQL> INPUT WHERE sid=103
SQL> RUN
  1  SELECT *
  2  FROM student
  3  WHERE sid=103
  4* ORDER BY name

       SID NAME                           ADDRESS
---------- ------------------------------ ------------------------------
       103 Hari                           Ktm
```

DEL Command

Deletes one more lines of the buffer.

Term	Description
n	Deletes line *n*.
n m	Deletes lines *n* through *m*.
n *	Deletes line *n* through the current line.
n LAST	Deletes line *n* through the last line.
*	Deletes the current line.
* *n*	Deletes the current line through line *n*.
* LAST	Deletes the current line through the last line.
LAST	Deletes the last line.

Syntax:
DEL n | n m | n* …

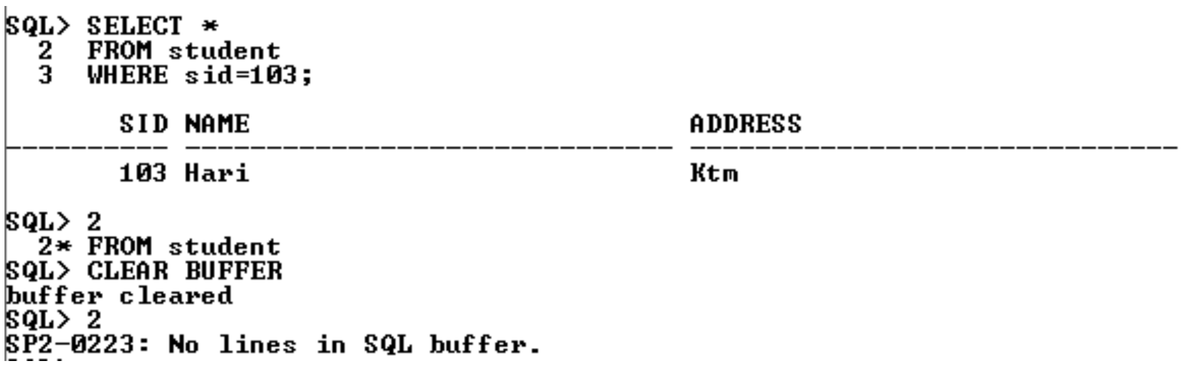


```
SQL> SELECT *
  2  FROM student
  3  WHERE sid=103;

       SID NAME                           ADDRESS
---------- ------------------------------ ------------------------------
       103 Hari                           Ktm

SQL> 3
  3* WHERE sid=103
SQL> DEL
SQL> RUN
  1  SELECT *
  2* FROM student

       SID NAME                           ADDRESS
---------- ------------------------------ ------------------------------
       101 Ram                            Btm
       102 Gita                           Btm
       103 Hari                           Ktm
```

CLEAR BUFFER Command

Clears text from the buffer.

Syntax:
CLEAR BUFFER



```
SQL> SELECT *
  2  FROM student
  3  WHERE sid=103;

       SID NAME                           ADDRESS
---------- ------------------------------ ------------------------------
       103 Hari                           Ktm

SQL> 2
  2* FROM student
SQL> CLEAR BUFFER
buffer cleared
SQL> 2
SP2-0223: No lines in SQL buffer.
```

Accepting Values at Runtime

To read the user input and store it in a variable, for later use, you can use SQL*Plus command ACCEPT.

Syntax:

```
ACCEPT <your variable> <variable type if needed [number|char|date]>
prompt 'message'
```

Example:

```
SQL> ACCEPT x NUMBER PROMPT 'Enter any number: '
Enter any number: 34
```

You can use this value later in PL/SQL.

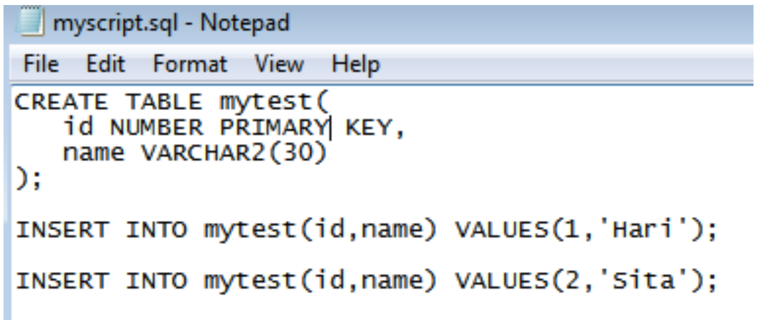
Prompting at Run Time while executing Query:

```
SQL> SELECT * FROM STUDENT
  2  WHERE sid=&sid;
Enter value for sid: 102
old   2: WHERE sid=&sid
new   2: WHERE sid=102

       SID NAME                           ADDRESS
---------- ------------------------------ ----------
       102 Gita                           Btm

SQL> SELECT * FROM STUDENT
  2  WHERE address='&address';
Enter value for address: Btm
old   2: WHERE address='&address'
new   2: WHERE address='Btm'

       SID NAME                           ADDRESS
---------- ------------------------------ ----------
       101 Ram                            Btm
       102 Gita                           Btm
```

Using Script Files

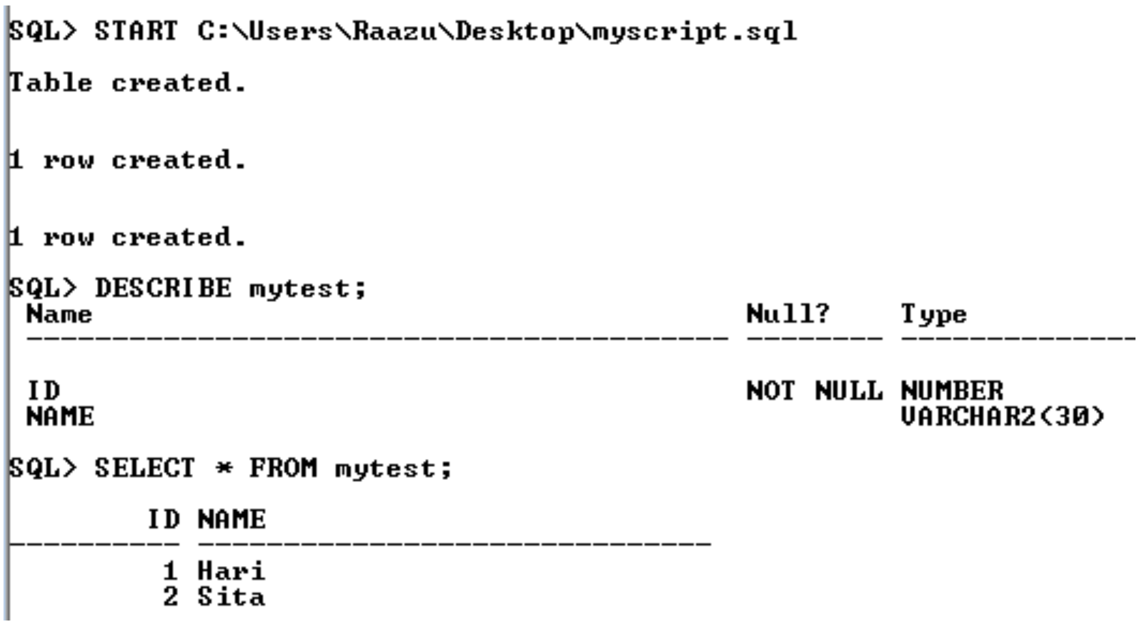
A SQL script is **a set of SQL commands saved as a file in** SQL Scripts. A SQL script can contain one or more SQL statements or PL/SQL blocks. You can use SQL Scripts to create, edit, view, run, and delete database objects.

At first create script file and save as follows:



```
CREATE TABLE mytest(
   id NUMBER PRIMARY KEY,
   name VARCHAR2(30)
);

INSERT INTO mytest(id,name) VALUES(1,'Hari');

INSERT INTO mytest(id,name) VALUES(2,'Sita');
```

Now, run script file using START command as follows:



```
SQL> START C:\Users\Raazu\Desktop\myscript.sql

Table created.


1 row created.


1 row created.

SQL> DESCRIBE mytest;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------

 ID                                        NOT NULL NUMBER
 NAME                                               VARCHAR2(30)

SQL> SELECT * FROM mytest;

        ID NAME
---------- ------------------------------
         1 Hari
         2 Sita
```

Oracle Data Types:

Oracle String data types

CHAR(size)	It is used to store character data within the predefined length. It can be stored up to 2000 bytes.
NCHAR(size)	It is used to store national character data within the predefined length. It can be stored up to 2000 bytes.
VARCHAR2(size)	It is used to store variable string data within the predefined length. It can be stored up to 4000 byte.
VARCHAR(SIZE)	It is the same as VARCHAR2(size). You can also use VARCHAR(size), but it is suggested to use VARCHAR2(size)
NVARCHAR2(size)	It is used to store Unicode string data within the predefined length. We have to must specify the size of NVARCHAR2 data type. It can be stored up to 4000 bytes.

Oracle Numeric Data Types

NUMBER(p, s)	It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.
FLOAT(p)	It is a subtype of the NUMBER data type. The precision p can range from 1 to 126.
BINARY_FLOAT	It is used for binary precision(32-bit). It requires 5 bytes, including length byte.
BINARY_DOUBLE	It is used for double binary precision (64-bit). It requires 9 bytes, including length byte.

Oracle Date and Time Data Types

DATE	It is used to store a valid date-time format with a fixed length. Its range varies from January 1, 4712 BC to December 31, 9999 AD.
TIMESTAMP	It is used to store the valid date in YYYY-MM-DD with time hh:mm:ss format.

Oracle Large Object Data Types (LOB Types)

BLOB	It is used to specify unstructured binary data. Its range goes up to 2^{32}-1 bytes or 4 GB.
BFILE	It is used to store binary data in an external file. Its range goes up to 2^{32}-1 bytes or 4 GB.
CLOB	It is used for single-byte character data. Its range goes up to 2^{32}-1 bytes or 4 GB.
NCLOB	It is used to specify single byte or fixed length multibyte national character set (NCHAR) data. Its range is up to 2^{32}-1 bytes or 4 GB.
RAW(size)	It is used to specify variable length raw binary data. Its range is up to 2000 bytes per row. Its maximum size must be specified.
LONG RAW	It is used to specify variable length raw binary data. Its range up to 2^{31}-1 bytes or 2 GB, per row.

Fundamental SQL Commands

Data Definition Language (DDL) Commands

Data Definition Language (DDL) Commands in Oracle are used to define the database objects such as Tables, Views, Stored Procedures, Stored Functions, Triggers, etc. That means DDL statements in Oracle are used to ALTER or Modify a database or table structure and schema. The most important point that you need to remember is DDL Commands in Oracle are working on the structure of a database object, not on the data of a table.

DDL contains five commands. They are as follows.

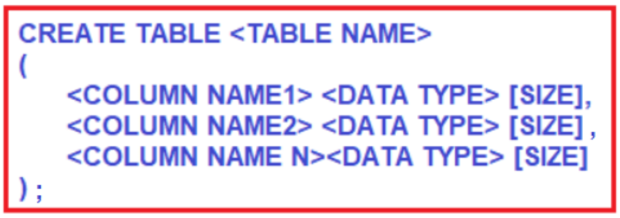
1. **Create**
2. **Alter**
3. **Rename**
4. **Truncate**
5. **Drop**

The following three DDL Commands are the latest feature in Oracle

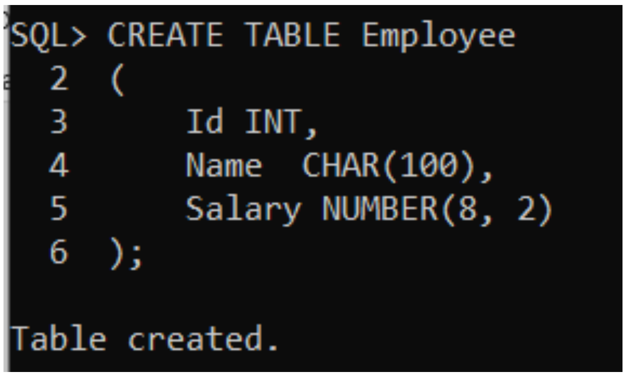
1. **Recyclebin**
2. **Flashback**
3. **Purge**

CREATE Command

The **CREATE** DDL command in the Oracle database is used to create a new database object such as Table, View, Stored Procedure, Stored Function, Trigger, etc.
The syntax to create a table is shown below.



```
CREATE TABLE <TABLE NAME>
(
    <COLUMN NAME1> <DATA TYPE> [SIZE],
    <COLUMN NAME2> <DATA TYPE> [SIZE] ,
    <COLUMN NAME N><DATA TYPE> [SIZE]
);
```



```
SQL> CREATE TABLE Employee
  2  (
  3      Id INT,
  4      Name  CHAR(100),
  5      Salary NUMBER(8, 2)
  6  );

Table created.
```

Rules for creating a table in Oracle:

1. The Table name must begin with a letter A-Z or a-z
2. The Table name can contain numbers and underscores
3. The name of the table can be in UPPER or lower case
4. The maximum length of the table name can be 30 characters
5. We cannot use the same name of another existing object in our schema
6. The table name must not be a SQL reserved word.
7. Don't provide space in the table name. If you want to provide space in a table name then you can use the underscore symbol.
8. A table should contain a minimum of 1 column and a maximum of 1000 columns

ALTER Command

This command is used to change or modify the structure of a database object i.e. Table, Views, Stored Procedures, Stored Functions, Triggers, etc.

In Oracle, using the ALTER DDL command we can perform the following operations on an existing table.

1. Increase/decrease the length of a column.
2. Change the data type of a column.
3. Change NOT NULL to NULL or NULL to NOT NULL.
4. Used to add a new column to an existing table.
5. Used to drop an existing column.
6. Add a new constraint.
7. Used to drop an existing constraint on a table.
8. To change the column name of a table.

To change/modify the structure of a table, we can use the following **sub-commands of Alter Data Definition Language (DDL) Command** in Oracle.

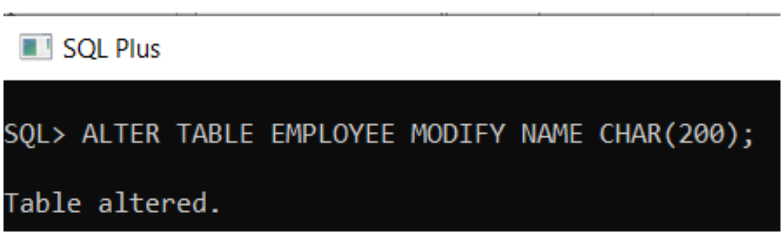
1. **ALTER – MODIFY**
2. **ALTER – ADD**
3. **ALTER – RENAME**
4. **ALTER – DROP**

ALTER – MODIFY Command

This command in the Oracle database is used to change a data type from an old data type to a new data type and also to change the size of the data type of a column.

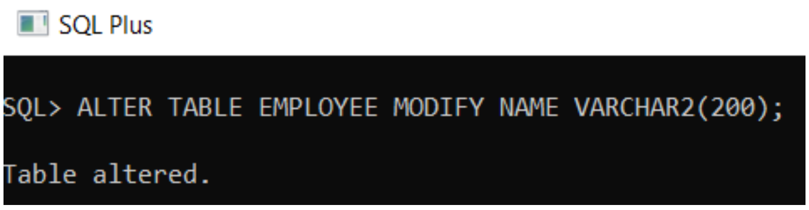
The syntax to use this ALTER – MODIFY command is given below.
ALTER TABLE <TABLE NAME> MODIFY <COLUMN NAME> <NEW DATATYPE> [NEW SIZE];

Changing size of data type:

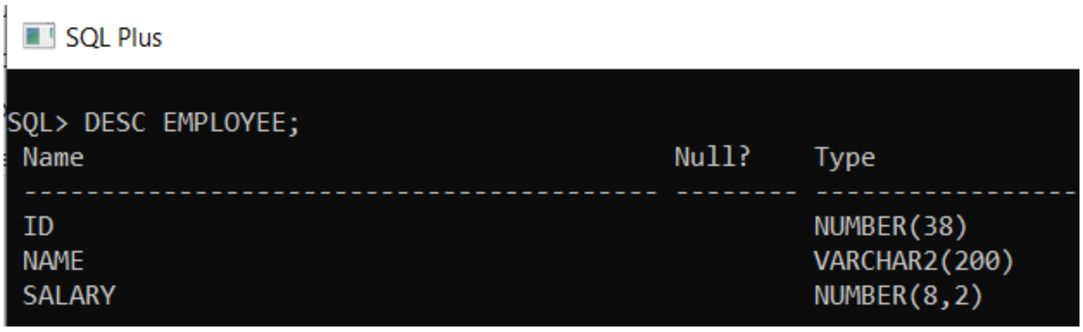


```
SQL> ALTER TABLE EMPLOYEE MODIFY NAME CHAR(200);

Table altered.
```

Changing data type:



```
SQL> ALTER TABLE EMPLOYEE MODIFY NAME VARCHAR2(200);

Table altered.
```



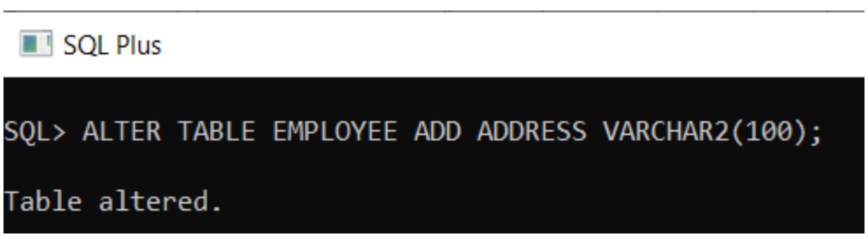
```
SQL> DESC EMPLOYEE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 ID                                                 NUMBER(38)
 NAME                                               VARCHAR2(200)
 SALARY                                             NUMBER(8,2)
```

<u>ALTER – ADD Command</u>

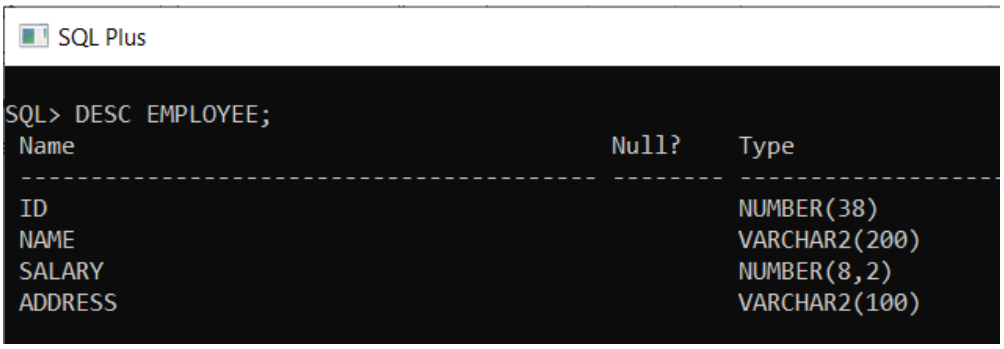
This command is used to add new column to existing table in Oracle.

Syntax:

ALTER TABLE <TABLE NAME> ADD <NEW COLUMN NAME> <DATATYPE>[SIZE];



```
SQL> ALTER TABLE EMPLOYEE ADD ADDRESS VARCHAR2(100);

Table altered.
```

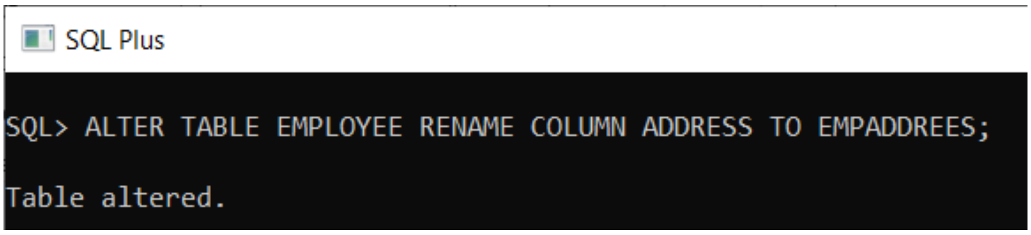


```
SQL> DESC EMPLOYEE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 ID                                                 NUMBER(38)
 NAME                                               VARCHAR2(200)
 SALARY                                             NUMBER(8,2)
 ADDRESS                                            VARCHAR2(100)
```

ALTER – RENAME Command

This command is used to change the column name in a table.

Syntax:
ALTER TABLE <TABLE NAME> RENAME <COLUMN> <OLD COLUMN NAME> TO <NEW COLUMN NAME>;

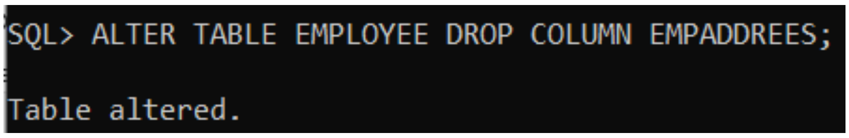


```
SQL> ALTER TABLE EMPLOYEE RENAME COLUMN ADDRESS TO EMPADDREES;

Table altered.
```

ALTER – DROP Command

This command is used to delete or drop a column from an existing table.

Syntax:
ALTER TABLE <TABLE NAME> DROP <COLUMN> <COLUMN NAME>;

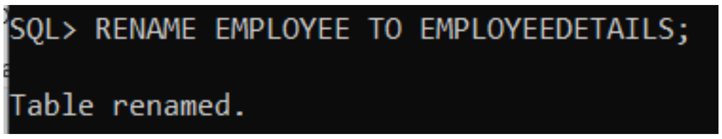


```
SQL> ALTER TABLE EMPLOYEE DROP COLUMN EMPADDREES;

Table altered.
```

RENAME DDL Command

This command is used to change the table name.

RENAME <OLD TABLE NAME> TO <NEW TABLE NAME>;



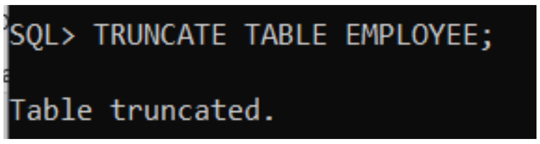
```
SQL> RENAME EMPLOYEE TO EMPLOYEEDETAILS;

Table renamed.
```

TRUNCATE DDL Command

If you want to delete all the records or rows from a table without any condition, then you need to use the Truncate DDL command in Oracle.

So, using this TRUNCATE DDL command you cannot delete specific records from the table because this command does not support the "WHERE" clause.

TRUNCATE TABLE <TABLE NAME>;



```
SQL> TRUNCATE TABLE EMPLOYEE;

Table truncated.
```

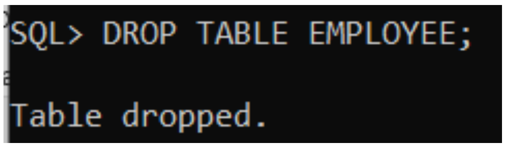
Points to Remember while working with TRUNCATE Command:

1. It is used to delete all rows from a table at a time.
2. It is used for deleting rows but not columns.
3. Rows are deleted permanently.
4. Cannot delete a specific row from a table.
5. It is not supporting the "where" condition.
6. The truncate command will delete rows but not the structure of the table.

DROP DDL Command

If you want to delete the table (rows & columns) from the database, then you need to use the DROP DDL command in oracle.

DROP TABLE <TABLE NAME>;



```
SQL> DROP TABLE EMPLOYEE;

Table dropped.
```

Note: Before oracle10g enterprise edition when we drop a table from the database, the table is permanently dropped. But, from oracle 10g enterprise edition once we drop a table from the database then it will drop temporarily. And the user has a chance to restore the dropped table again into the database by using the following commands are,

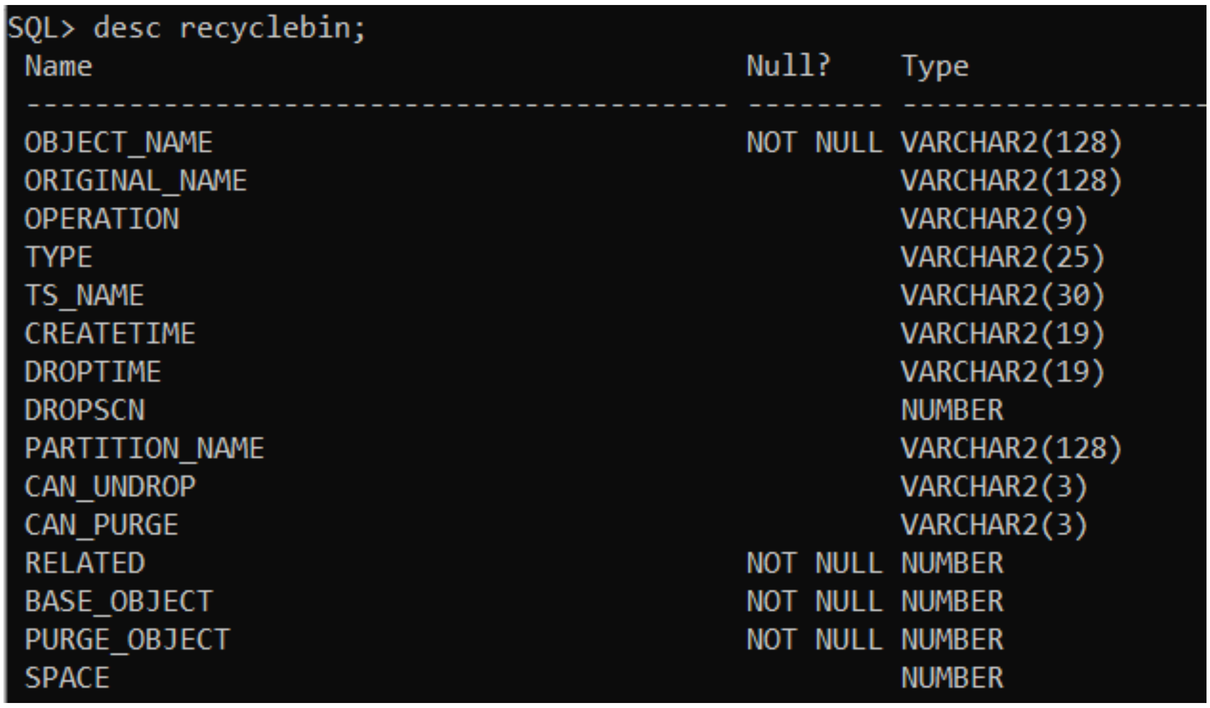
1. **Recyclebin**
2. **Flashback**
3. **Purge**

RECYCLEBIN Command

It is a system-defined table that is used for storing the information about the dropped tables. It is similar to Windows Recyclebin.

To view the structure of recycle bin, you can use the following syntax.

DESC RECYCLEBIN;

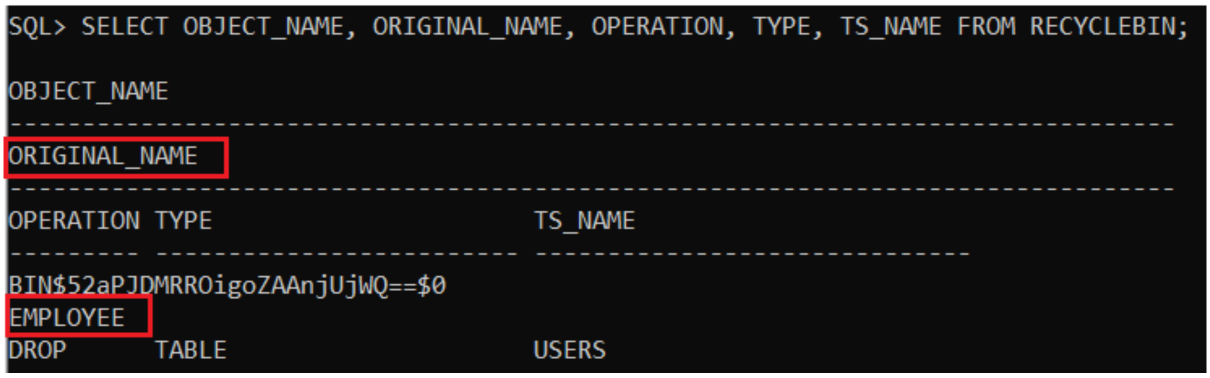


```
SQL> desc recyclebin;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 OBJECT_NAME                               NOT NULL VARCHAR2(128)
 ORIGINAL_NAME                                      VARCHAR2(128)
 OPERATION                                          VARCHAR2(9)
 TYPE                                               VARCHAR2(25)
 TS_NAME                                            VARCHAR2(30)
 CREATETIME                                         VARCHAR2(19)
 DROPTIME                                           VARCHAR2(19)
 DROPSCN                                            NUMBER
 PARTITION_NAME                                     VARCHAR2(128)
 CAN_UNDROP                                         VARCHAR2(3)
 CAN_PURGE                                          VARCHAR2(3)
 RELATED                                   NOT NULL NUMBER
 BASE_OBJECT                               NOT NULL NUMBER
 PURGE_OBJECT                              NOT NULL NUMBER
 SPACE                                              NUMBER
```

How to view dropped tables in recycle bin?

To view the dropped tables in recycle bin we need to query the recyclebin as follows. You can specify the column names that are available in the recyclebin table.

SELECT OBJECT_NAME, ORIGINAL_NAME, OPERATION, TYPE, TS_NAME FROM RECYCLEBIN;

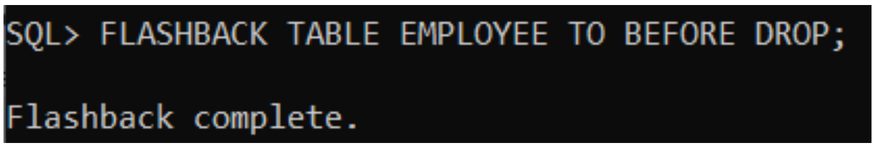


```
SQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OPERATION, TYPE, TS_NAME FROM RECYCLEBIN;

OBJECT_NAME
--------------------------------------------------------------------------------
ORIGINAL_NAME
--------------------------------------------------------------------------------
OPERATION TYPE                      TS_NAME
--------- ------------------------- ------------------------------
BIN$52aPJDMRROigoZAAnjUjWQ==$0
EMPLOYEE
DROP      TABLE                     USERS
```

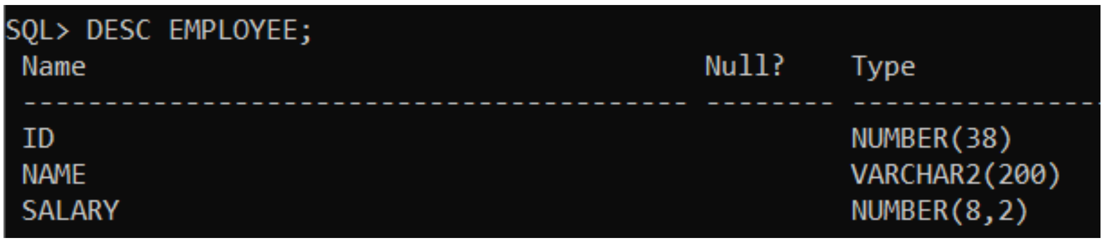
FLASHBACK Command

The Flashback command is used to restore the dropped table from recycle bin to the database.

FLASHBACK TABLE <TABLE NAME> TO BEFORE DROP;



```
SQL> FLASHBACK TABLE EMPLOYEE TO BEFORE DROP;

Flashback complete.
```

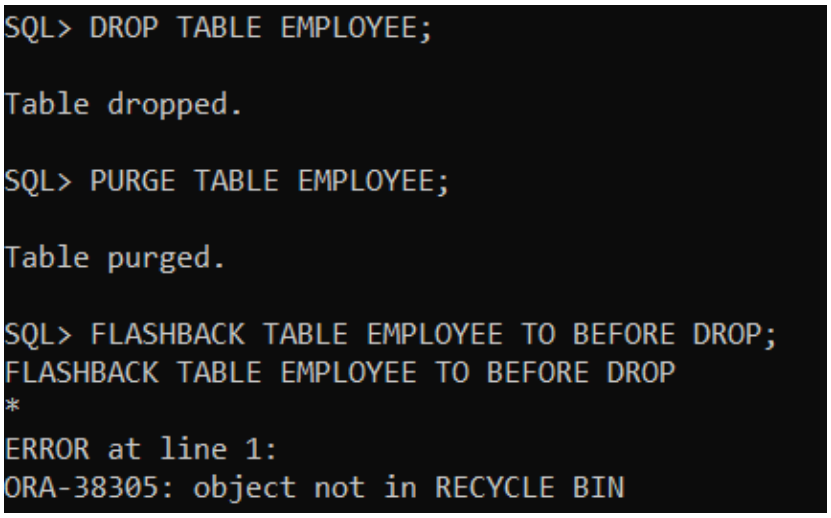


```
SQL> DESC EMPLOYEE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 ID                                                 NUMBER(38)
 NAME                                               VARCHAR2(200)
 SALARY                                             NUMBER(8,2)
```

PURGE Command

This DDL command is used to drop a table permanently i.e. it will drop a specific table from recycle bin permanently.

PURGE TABLE <TABLE NAME>;



```
SQL> DROP TABLE EMPLOYEE;

Table dropped.

SQL> PURGE TABLE EMPLOYEE;

Table purged.

SQL> FLASHBACK TABLE EMPLOYEE TO BEFORE DROP;
FLASHBACK TABLE EMPLOYEE TO BEFORE DROP
*
ERROR at line 1:
ORA-38305: object not in RECYCLE BIN
```

How to drop all tables from recycle bin permanently in Oracle?

To drop all tables from the recycle bin permanently we need to use the following syntax.

PURGE RECYCLEBIN;

How to drop a table from the database permanently?

To drop a table permanently from the database in oracle we need to use the following syntax.

DROP TABLE <TABLE NAME> PURGE;

Example: **DROP TABLE EMPLOYEE PURGE;**

Data Manipulation Language(DML) Commands

DML commands are basically used to INSERT, UPDATE, and DELETE data in a database table. That means DML statements affect the records in a table. These are the basic operations that we perform on data such as inserting new records, deleting unnecessary records, and updating/modifying existing records.

DML provides the following three commands:

1. **INSERT** – To insert new records
2. **UPDATE** – To update/Modify existing records
3. **DELETE** – To delete existing records

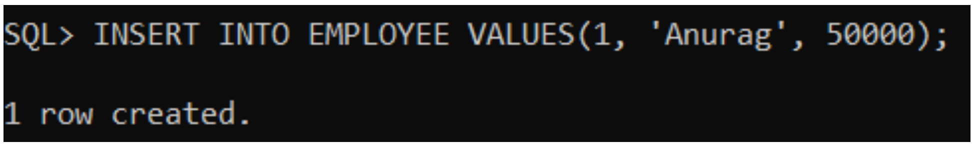
The following two are the Latest DML Commands in the Oracle database.

1. **INSERT ALL**
2. **MERGE**

INSERT DML Command

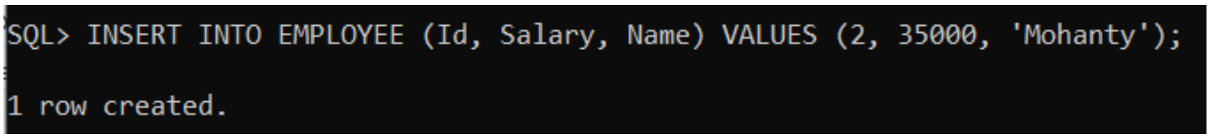
The INSERT DML command is used to insert a new row into a table.

INSERT INTO <TableName> VALUES(V1,V2,V3,……….);



```
SQL> INSERT INTO EMPLOYEE VALUES(1, 'Anurag', 50000);

1 row created.
```

INSERT INTO <Table Name> (REQUIRED COLUMN NAMES) VALUES (V1, V2,……..);



```
SQL> INSERT INTO EMPLOYEE (Id, Salary, Name) VALUES (2, 35000, 'Mohanty');

1 row created.
```

How to insert NULLs into a table?

Method1:
INSERT INTO EMPLOYEE VALUES (NULL, NULL, NULL);

Method2:
INSERT INTO EMPLOYEE (Id, Salary, Name) VALUES (NULL, NULL, NULL);

How to insert multiple rows into a table?

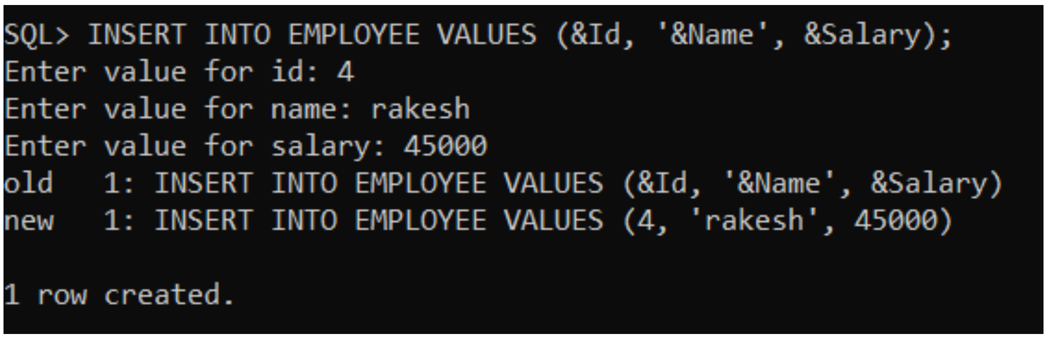
We need to use the following two substitutional operators:

1. &: to insert values to columns dynamically (we can change values)
2. &&: to insert fixed values to columns (we cannot change values). We can insert values to columns in a fixed manner. If we want to change a fixed value of a column then we should "exit" from the oracle database.

Syntax1(&): For all columns

The following is the syntax to use & to insert values to columns dynamically for all the columns of a table.

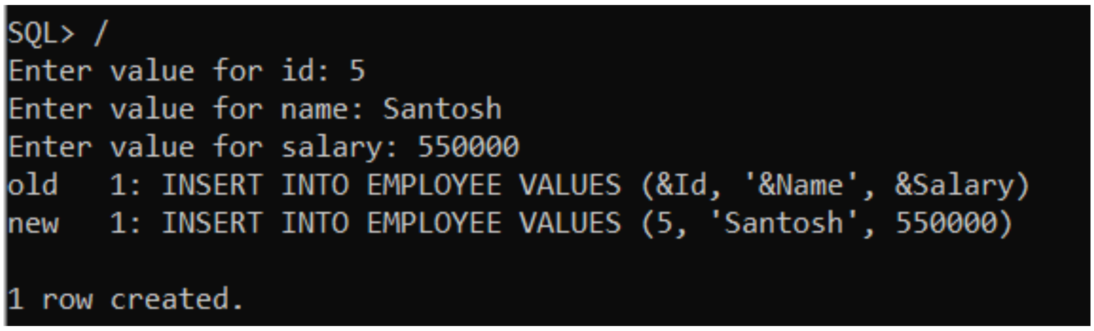
INSERT INTO <TABLE NAME> VALUES(&<COL1>, &<COL2>,………….);



```
SQL> INSERT INTO EMPLOYEE VALUES (&Id, '&Name', &Salary);
Enter value for id: 4
Enter value for name: rakesh
Enter value for salary: 45000
old   1: INSERT INTO EMPLOYEE VALUES (&Id, '&Name', &Salary)
new   1: INSERT INTO EMPLOYEE VALUES (4, 'rakesh', 45000)

1 row created.
```

If you want to insert another row into the Employee table, then simply type / and press the enter key in the SQL Plus editor.

The / is used to re-execute the last executed SQL query in SQL Plus editor. Once you type the / button and press the enter key then again it will ask you to enter the Id, Name, and salary column values one by one as shown in the below image.

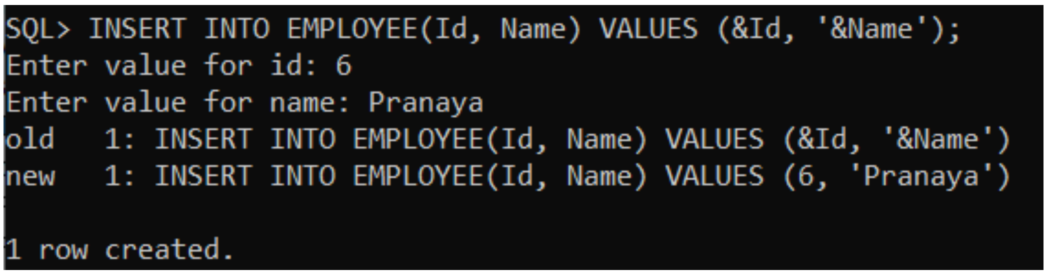


```
SQL> /
Enter value for id: 5
Enter value for name: Santosh
Enter value for salary: 550000
old   1: INSERT INTO EMPLOYEE VALUES (&Id, '&Name', &Salary)
new   1: INSERT INTO EMPLOYEE VALUES (5, 'Santosh', 550000)

1 row created.
```

Syntax2(&): For required columns

The following is the syntax to use & to insert values to columns dynamically for the required columns of a table.

INSERT INTO <TABLE NAME >(REQUIRED COLUMN NAMES) VALUES (&<COL1>,&<COL2>,…….);



```
SQL> INSERT INTO EMPLOYEE(Id, Name) VALUES (&Id, '&Name');
Enter value for id: 6
Enter value for name: Pranaya
old   1: INSERT INTO EMPLOYEE(Id, Name) VALUES (&Id, '&Name')
new   1: INSERT INTO EMPLOYEE(Id, Name) VALUES (6, 'Pranaya')

1 row created.
```

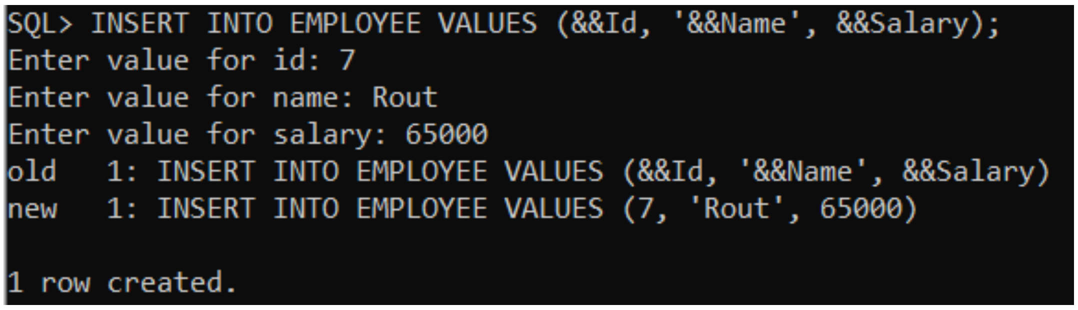
If you want to insert another row into the Employee table with the Id and Name column values, then simply type / and press the enter button in the SQL Plus editor.

Like in previous example.

Syntax1(&&): For all Columns

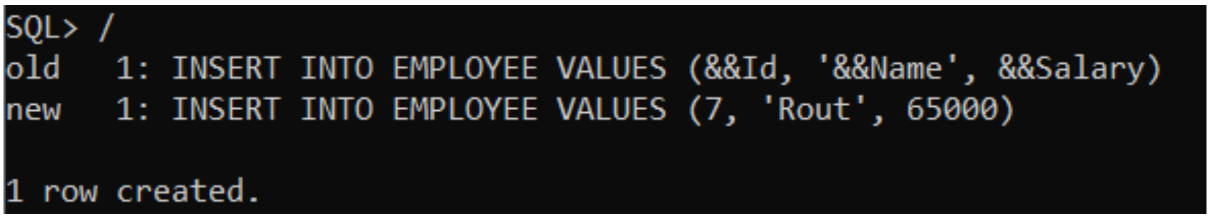
The following is the syntax to use "&&" to insert fixed values to columns for all the columns of a table. In this case, we cannot change values.

INSERT INTO <TABLE NAME> VALUES(&&<COL1>, &&<COL2>,………….);



```
SQL> INSERT INTO EMPLOYEE VALUES (&&Id, '&&Name', &&Salary);
Enter value for id: 7
Enter value for name: Rout
Enter value for salary: 65000
old   1: INSERT INTO EMPLOYEE VALUES (&&Id, '&&Name', &&Salary)
new   1: INSERT INTO EMPLOYEE VALUES (7, 'Rout', 65000)

1 row created.
```

Now, if you type / and press the enter key, then it will insert the same record once more into the Employee table as shown in the below image. That means it is going to insert fixed values. The value cannot be changed dynamically.



```
SQL> /
old   1: INSERT INTO EMPLOYEE VALUES (&&Id, '&&Name', &&Salary)
new   1: INSERT INTO EMPLOYEE VALUES (7, 'Rout', 65000)

1 row created.
```

Syntax2(&&): For Required Columns

INSERT INTO <TABLE NAME >(REQUIRED COLUMN NAMES) VALUES (&&<COL1>,&&<COL2>,…….);

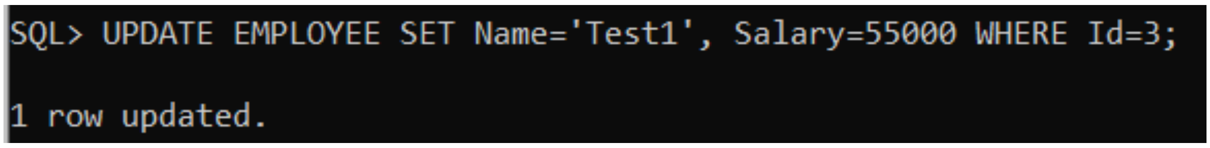
Example:
Do Yourself.

UPDATE DML Command

The UPDATE DML Statement in Oracle is basically used to

1. To update all rows data in a table at a time.
2. To update a specific row data in a table by using the "where" condition.

UPDATE <TABLE NAME> SET <COLUMN NAME1>=<VALUE1>, <COLUMN NAME2>=<VALUE2>, ……..[WHERE <CONDITION>];



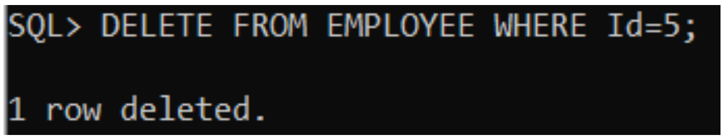
```
SQL> UPDATE EMPLOYEE SET Name='Test1', Salary=55000 WHERE Id=3;

1 row updated.
```

DELETE DML Command

The DELETE DML Statement in Oracle is basically used to

1. To delete all rows from a table at a time.
2. To delete a specific row from a table by using the “where” condition.

DELETE FROM <TABLE NAME> [WHERE <CONDITION>];



```
SQL> DELETE FROM EMPLOYEE WHERE Id=5;

1 row deleted.
```

Delete VS Truncate in Oracle:

Delete

1. It is a DML command
2. It can delete a specific row from a table.
3. It supports the “WHERE” clause condition.
4. It is a temporary data deletion.
5. We can restore deleted data by using the “ROLLBACK” command.
6. Execution speed is slow as deleting operation is performed row by row / one by one manner.

Truncate

1. It is a DDL command
2. It is not possible to delete a specific row from a table.
3. It does not support the “WHERE” clause condition.
4. It is a permanent data deletion.
5. We cannot restore deleted data by using the “ROLLBACK” command.
6. Execution speed is fast as deleting a group of rows at a time.

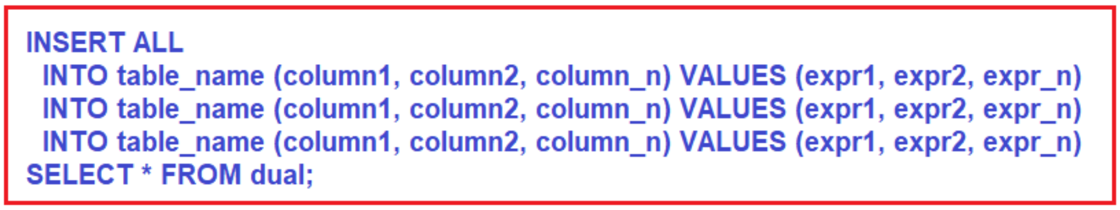
New DML Commands in Oracle

The following are New DML Commands introduced in Oracle for DML operations:

1. **Insert all**
2. **Merge**

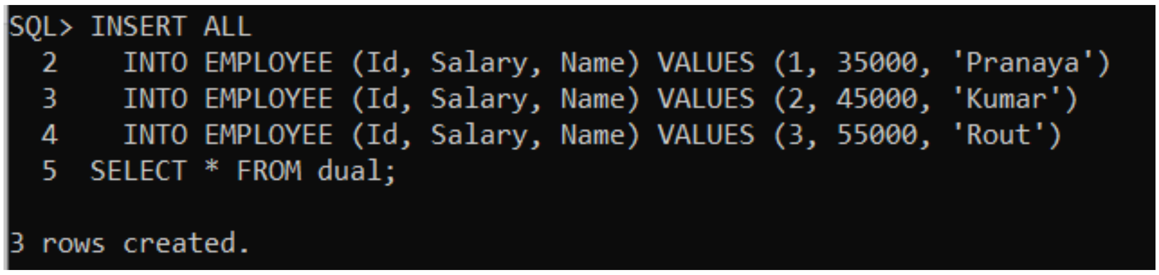
INSERT ALL Command

The INSERT ALL statement in Oracle is basically used to add multiple rows with a single INSERT statement. The important point that you need to remember is, the rows can be inserted into one table or multiple tables using only one SQL command.



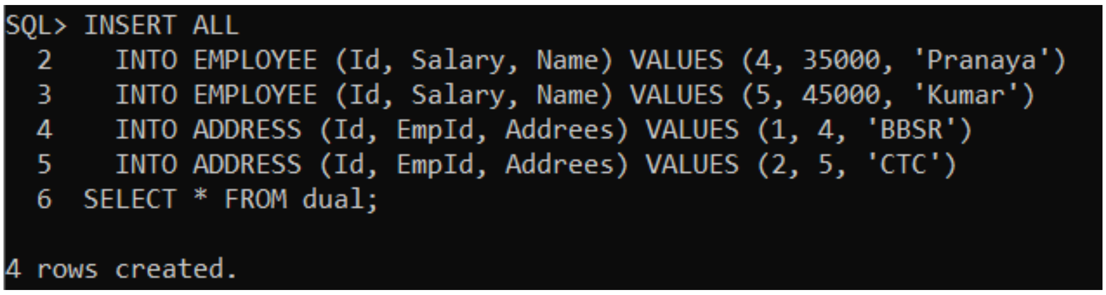
```
INSERT ALL
  INTO table_name (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO table_name (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
  INTO table_name (column1, column2, column_n) VALUES (expr1, expr2, expr_n)
SELECT * FROM dual;
```

Example: Insert into One Table



```
SQL> INSERT ALL
  2    INTO EMPLOYEE (Id, Salary, Name) VALUES (1, 35000, 'Pranaya')
  3    INTO EMPLOYEE (Id, Salary, Name) VALUES (2, 45000, 'Kumar')
  4    INTO EMPLOYEE (Id, Salary, Name) VALUES (3, 55000, 'Rout')
  5  SELECT * FROM dual;

3 rows created.
```

Example: Insert into Multiple Table



```
SQL> INSERT ALL
  2    INTO EMPLOYEE (Id, Salary, Name) VALUES (4, 35000, 'Pranaya')
  3    INTO EMPLOYEE (Id, Salary, Name) VALUES (5, 45000, 'Kumar')
  4    INTO ADDRESS (Id, EmpId, Addrees) VALUES (1, 4, 'BBSR')
  5    INTO ADDRESS (Id, EmpId, Addrees) VALUES (2, 5, 'CTC')
  6  SELECT * FROM dual;

4 rows created.
```

Merge DML Command

Will be discussed later….

Data Query language (DQL) / Data Retrieve Language (DRL)

The DQL statements in Oracle or you can say the **SELECT** statements. The Select Statement in Oracle is basically used to return records in the form of a result set from one or more tables or views.

The SQL Select Query does not store any data itself. It simply displays the data that is stored in database tables. The Select Statement in Oracle can retrieve and shows the data from one or more database tables or views, from other queries, or from a combination of the above two.

Example: Retrieve all fields from the Employee table

SELECT * FROM Employee;

Example: Retrieve Specific Columns from Employee table

SELECT Id, Name, Department, Salary FROM Employee;

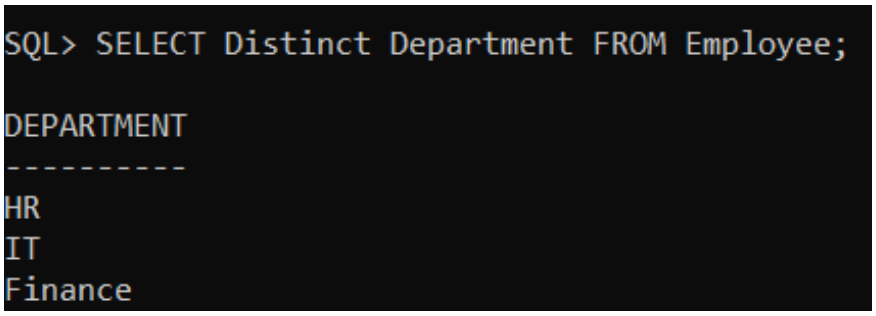
Example: Retrieve all Columns with Condition in Oracle

SELECT * FROM Employee WHERE Gender = ‘Male’;

SELECT DISTINCT Statement in Oracle

The SELECT DISTINCT SQL statement in Oracle is used to return only the distinct or different values from a table column. In a database table, a column may contain duplicate or similar values.

SELECT Distinct Department FROM Employee;



```
SQL> SELECT Distinct Department FROM Employee;

DEPARTMENT
----------
HR
IT
Finance
```

Alias Names in Oracle:

It is nothing but an alternate (or) temporary name. Users can create alias names on two levels in a database.

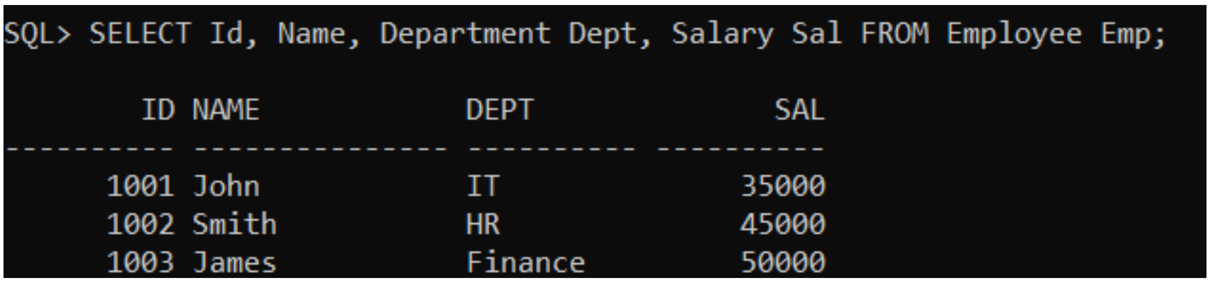
I) Column Level:
At this level, we are creating alias names on columns. The syntax is given below.
<column name> <column alias name>
Example: Department Dept

II) Table Level:
In this level, we create alias names on the table. The syntax is given below.
<table name> <table alias name>
Example: Employee Emp

Syntax to combined column + table level alias names by using "select" query:

Select <column name1> <column name1 alias name>, <column name2> <column name2 alias name> From <Table Name> <Table Alias Name>;

Example:

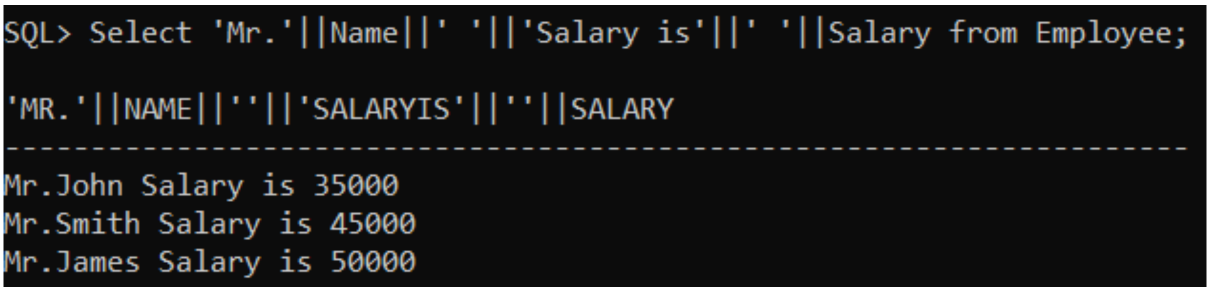


```
SQL> SELECT Id, Name, Department Dept, Salary Sal FROM Employee Emp;

        ID NAME            DEPT              SAL
---------- --------------- ---------- ----------
      1001 John            IT              35000
      1002 Smith           HR              45000
      1003 James           Finance         50000
```

Concatenation Operator (||) in Oracle:

This operator is used to join two string values (or) two expressions in a select query.

Example: Select 'Mr.'||Name||' '||'Salary is'||' '||Salary from Employee;
When you execute the above query, you will get the following output.

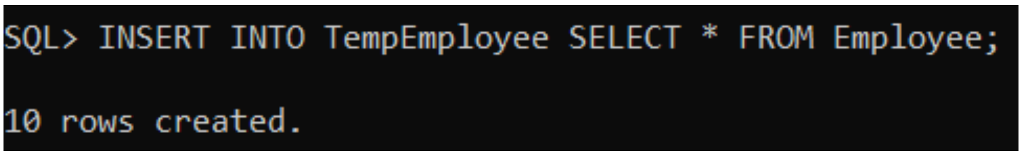


```
SQL> Select 'Mr.'||Name||' '||'Salary is'||' '||Salary from Employee;

'MR.'||NAME||''||'SALARYIS'||''||SALARY
--------------------------------------------------------------------
Mr.John Salary is 35000
Mr.Smith Salary is 45000
Mr.James Salary is 50000
```

How to copy data from one table to another table in Oracle?

Example 1
Suppose we have created the new table and then we want to insert data from the Existing table to the new table.

INSERT INTO <DESTINATION TABLE NAME> SELECT * FROM <SOURCE TABLE NAME>;

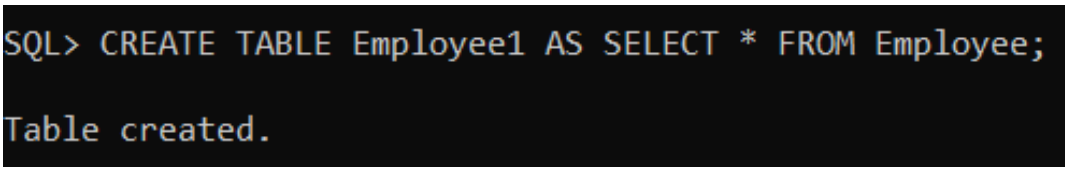


```
SQL> INSERT INTO TempEmployee SELECT * FROM Employee;

10 rows created.
```

Example 2
Suppose you want to create a new table with the same structure and same data as the existing table. For example, we want to create a new table with the name Employee1 with the same structure and same data as the Employee table, then we need to execute the following SQL query.

CREATE TABLE Employee1 AS SELECT * FROM Employee;

When you execute the above query, you will get the following output.

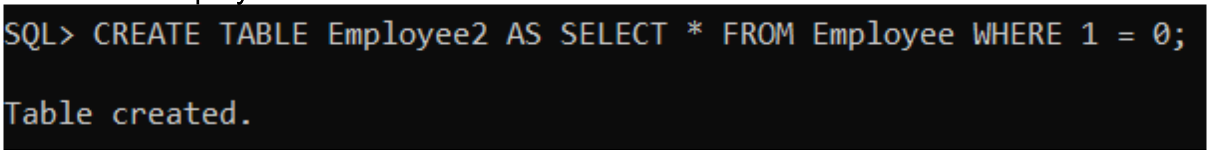


```
SQL> CREATE TABLE Employee1 AS SELECT * FROM Employee;

Table created.
```

Example 3
If you want to create a new table with the same structure as an existing table but without data. For example, we want to create a new table with the name Employee2 with the same structure as the existing Employee table but with output data. To do so, we need to execute the following SQL query.

CREATE TABLE Employee2 AS SELECT * FROM Employee WHERE 1 = 0;

When you execute the above SQL query, you will get the following output. Here, it will create the Employee2 table without data.



```
SQL> CREATE TABLE Employee2 AS SELECT * FROM Employee WHERE 1 = 0;

Table created.
```

Merge Command in Oracle:

It is a DML command. It is used to transfer data from the source table to the destination table.

- **If data is matching in both tables**, then those matching data /rows are overridden on the destination table by using the "**UPDATE**" command
- **whereas if data is not matching** then those unmatching data/rows are transferred from the source table to the destination table by using the "**INSERT**" command.

```
MERGE INTO <DESTINATION TABLE NAME> <ALIAS NAME> USING <SOURCE TABLE NAME> <ALIAS NAME> ON
(<JOIINING CONDITION>)

WHEN MATCHED THEN
UPDATE SET <DEST.TABLE ALIAS NAME> <COLUMN NAME1>=<SOUR.TABLE ALIAS NAME> <COLUMN NAME1>

WHEN NOT MATCHED THEN
INSERT (<DESTINATION TABLE COLUMNS>) VALUES (<SOURCE TABLE COLUMNS>);
```

Example: Using MERGE Command in Oracle

Step1:
SELECT * FROM Employee;

Step2: Create a new table from the existing Employee table with data
CREATE TABLE Employee3 AS SELECT * FROM Employee;

Step3: Insert two more records into the new Employee3 table
INSERT INTO Employee3 (Id, Name, Department, Salary, Gender, Age, City) VALUES (1011, 'Kumar', 'IT', 55000, 'Male', 27, 'London');
INSERT INTO Employee3 (Id, Name, Department, Salary, Gender, Age, City) VALUES (1012, 'Santosh', 'HR', 70000, 'Female', 29, 'Mumbai');

Step4: Checking the data of both source and destination tables
SELECT * FROM Employee3; Source Table
SELECT * FROM Employee; Old Table

Step5: Using Merge statement

```
MERGE INTO Employee d using Employee3 s on(d.Id = s.Id)
when matched then
update set d.Name=s.Name, d.City=s.City,d.Department=s.Department,
d.Salary=s.Salary, d.Age=s.Age, d.Gender=s.Gender
when not matched then
Insert(d.ID, d.Name, d.Department, d.City, d.Salary, d.Age, d.Gender)values(s.Id,
s.Name, s.Department, s.City, s.Salary, s.Age, s.Gender);
```

Operators in Oracle

Operators are used to expressing the conditions in Select statements. The operator manipulates individual data items and returns a result. The data items are called operands or arguments.

The different types of Operators available in Oracle are as follows:

- **Arithmetic operators**
- **Assignment operator**
- **Relational operators**
- **Logical operators**
- **Special Operators**
- **Set Operators**

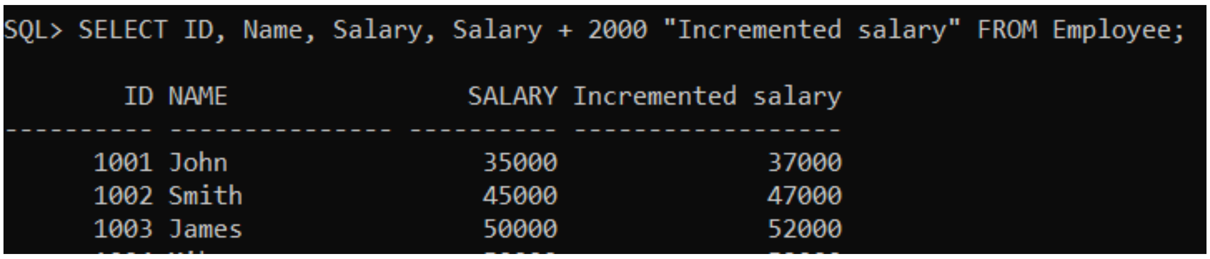
Arithmetic Operators in Oracle:

The Arithmetic Operators in Oracle are used for performing mathematical calculations such as Addition, Subtraction, Multiplication, Module, and Division represented by the expected +, -, *(star or asterisk), %, and / forward slash symbols respectively on the given operand values.

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b
-	Subtraction - Subtracts right hand operand from left hand operand	a - b
*	Multiplication - Multiplies values on either side of the operator	a * b
/	Division - Divides left hand operand by right hand operand	b / a
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a

Example: Display salary of employees with 2000 increments in their salary.

SELECT ID, Name, Salary, Salary + 2000 “Incremented salary” FROM Employee;



```
SQL> SELECT ID, Name, Salary, Salary + 2000 "Incremented salary" FROM Employee;

        ID NAME                SALARY Incremented salary
---------- --------------- ---------- ------------------
      1001 John                 35000              37000
      1002 Smith                45000              47000
      1003 James                50000              52000
```

Assignment Operator in Oracle

The equal sign (=) is the assignment operator where the value on the right is assigned to the value on the left.

Example: Increase salary by 1000

UPDATE Employee SET Salary = Salary + 1000;

Example: Increase salary by 15% whose id is 1006

UPDATE Employee SET Salary = Salary * 0.15 WHERE ID = 1006;

This operator is used for the equality test. Used to test the equality of two operands. For example, display the details of Employees whose Gender is Male.

SELECT * FROM Employee WHERE Gender = 'Male';

Comparison/Relational Operators in Oracle

As the name suggests the Relational/Comparison Operators in Oracle are used to compare two values i.e. these operators are used for comparing one expression with another expression. The relational operators determine whether the two values are equal or a value is greater than the other, or less than the other. The result of a comparison can be TRUE, FALSE, or NULL (When one or both the expressions contain NULL values).

The different types of relational operators that are available in Oracle are as follows:

1. **Equal (=) Operator**
2. **Not Equal (!= or <>) Operator**
3. **Greater Than (>) Operator**
4. **Less Than (<) Operator**
5. **Greater Than or Equal To (>=) Operator**
6. **Less Than or Equal To (<=) Operator**

Equal (=) Relational Operator in Oracle

SELECT * FROM Employee WHERE Gender = 'Male';

Not Equal (!=) Relational Operator in Oracle

SELECT * FROM Employee WHERE Gender != 'Male';

Not Equal (<>) Relational Operator in Oracle

The Not Equal (<>) Operator in Oracle is the same as the Not Equal (!=) operator. That means it is also used to check whether the two expressions are equal or not.

```
SELECT * FROM Employee WHERE Gender <> ‘Male’;
```

Greater Than (>) Relational Operator in Oracle

```
SELECT * FROM Employee WHERE Salary > 45000;
```

Less Than (<) Relational Operator in Oracle

```
SELECT * FROM Employee WHERE Salary < 50000;
```

Greater Than or Equal To (>=) Operator in Oracle

```
SELECT * FROM Employee WHERE Salary >= 50000;
```

Less Than or Equal To (<=) Operator in Oracle

```
SELECT * FROM Employee WHERE Salary <= 50000;
```

Logical Operators in Oracle

If you want to combine more than one condition, then you need to use the Logical Operators in Oracle. The Logical Operators in Oracle are basically used to check for the truth-ness of some conditions. Logical operators return a Boolean data type with a value of TRUE, or FALSE.

In Oracle, there are three Logical Operators available. They are as follows:

1. **AND**: TRUE if both Boolean expressions are TRUE.
2. **OR**: TRUE if one of the Boolean expressions is TRUE.
3. **NOT**: Reverses the value of any other Boolean operator.

The Logical Operators in Oracle are used to compare two conditions to check whether a row (or rows) can be selected for the output.

```
SELECT * FROM Employee WHERE Department = ‘IT’ AND Age = 26;

SELECT * FROM employee WHERE department = ‘IT’ AND NOT Age = 28;

SELECT * FROM Employee WHERE Age = 10 OR Age = 15;

SELECT * FROM Employee WHERE NOT City = ‘London’;
```

Nested Logical Operators in Oracle:

We can also use multiple logical operators in a single SQL statement in Oracle. When we combine the logical operators in a SELECT statement, the order in which the statement is processed is

1. **NOT**
2. **AND**
3. **OR**

```
SELECT * FROM Employee WHERE Salary >= 27000 AND Salary <= 30000 OR NOT CITY
= 'Mumbai';
```

In this case, the filter works as follows:

1. First, all the Employees who do not belong to the City of Mumbai are selected.
2. Second, all the Employees, whose Salary between 27000 and 30000 are selected.
3. And finally, the result is the rows that satisfy at least one of the above conditions are returned.

IN Operator in Oracle

The IN Operator in Oracle is used to search for specified values that match any value in the set of multiple values it accepts. The "IN" operator evaluates multiple values on a single data column. It displays the data row if any one of the given values is matched with the data column value. If none of the values matches, the SQL statement won't return that data row.

```
SELECT * FROM Employee WHERE Department IN ('IT', 'HR');
```

Once you execute the above SELECT SQL Query, then you will get the result set which includes only the IT and HR departments employees.

Advantages of IN Operator over OR Operator in Oracle:

Although both IN and OR going to provide the same results, **IN condition is more preferable because it has the minimum number of codes as compared to OR condition**.

For example, the following SQL query returns all the employees who belong to the IT or HR department using the Oracle OR Operator.

```
SELECT * FROM Employee WHERE (Department = 'IT' OR Department = 'HR');
```

The following SQL Query also returns all the employees who belong to the IT or HR department using the Oracle IN Operator.

```
SELECT * FROM Employee WHERE Department IN ('IT', 'HR');
```

Example: Fetch all the employees whose age is either 25 or 26

```
SELECT * FROM Employee WHERE Age IN (25, 26);
```

In Operator Inside DML statements:

UPDATE Employee SET Salary=Salary+200 WHERE ID IN (1002, 1004, 1006);

DELETE FROM Employee WHERE ID IN (1003, 1005);

BETWEEN Operator in Oracle:

The BETWEEN Operator in Oracle is used to get the values within a specified range.

If you want to fetch all the employees from the Employee table where the employee age is between 25 and 27, then we need to write the SELECT SQL Query using the Oracle Between Operator as shown below.

SELECT * FROM Employee WHERE Age BETWEEN 25 AND 27;

Once you execute the above SELECT Query, then you will get the following result set which includes only the employees whose age is between 25 and 27.

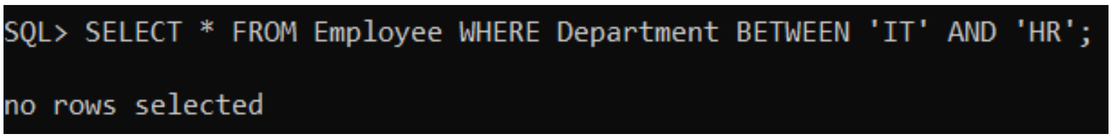
NOT BETWEEN Operator Example in Oracle:

SELECT * FROM Employee WHERE Age NOT BETWEEN 25 AND 27;

Between Operator with String Data Type in Oracle:

If you use the BETWEEN Operator with other data types except for numbers and date, then also you will get No Rows Selected as output.

In the below query, we are using Between Operator with string data type and hence we will get No Rows Selected as output.



```
SQL> SELECT * FROM Employee WHERE Department BETWEEN 'IT' AND 'HR';

no rows selected
```

LIKE Operator in Oracle

The LIKE Operator in Oracle is used with the WHERE clause to search for a specific pattern in a given string. The string pattern contains wildcard characters that represent missing characters.

SELECT * FROM Employee WHERE name LIKE ‘%P’;
Here the symbol percentage is a wildcard character.

Example: Fetch all the Employees whose names start with the letter P.
SELECT * FROM Employee WHERE Name LIKE ‘P%’;

Example: Fetch all the employees whose city ends with ai.
SELECT * FROM Employee WHERE City LIKE ‘%ai’;

Example: Fetch all employee whose name contains the word am
SELECT * FROM Employee WHERE Name LIKE '%am%';

NOT LIKE Operator in Oracle

Example: Fetch all the employees whose name does not start with P.
SELECT * FROM Employee WHERE Name NOT LIKE 'P%';

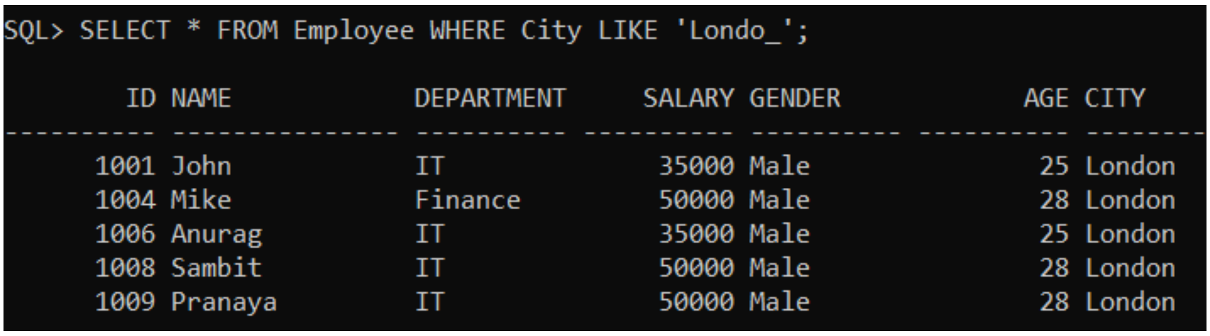
Example: Fetch all the employees whose city does not end with ai.
SELECT * FROM Employee WHERE CITY NOT LIKE '%ai%';

Example: Fetch all employee whose name does not contain the word am
SELECT * FROM Employee WHERE Name NOT LIKE '%am%';

Like Operator Using _ (underscore) Wildcard Character in Oracle

The underscore (_) means it will exactly search for one character after the "_".

SELECT * FROM Employee WHERE City LIKE 'Londo_';



```
SQL> SELECT * FROM Employee WHERE City LIKE 'Londo_';

        ID NAME            DEPARTMENT     SALARY GENDER            AGE CITY
---------- --------------- ---------- ---------- ---------- ---------- --------
      1001 John            IT              35000 Male               25 London
      1004 Mike            Finance         50000 Male               28 London
      1006 Anurag          IT              35000 Male               25 London
      1008 Sambit          IT              50000 Male               28 London
      1009 Pranaya         IT              50000 Male               28 London
```

Example: Fetch all the employees whose department is having two letters
Here we are using two underscores (__) as the part in the LIKE Operator.
SELECT * FROM Employee WHERE Department LIKE '__';

Example: Display all the employees whose names are having the second letter as 'i' in the Employee table
SELECT * FROM Employee WHERE Name LIKE '_i%';

SET Operators in Oracle

SQL set operators allow combining results from two or more SELECT statements.

There are four types of SET Operators available in Oracle. They are as follows:

1. **UNION**: It Returns all distinct rows selected by either query
2. **UNION ALL:** It Returns all rows selected by either query, including all duplicates
3. **INTERSECT**: It Returns all distinct rows selected by both queries
4. **MINUS**: It Returns all distinct rows selected by the first query but not the second

Set Operator Guidelines in Oracle:

1. In every result set the data type of each column must be compatible (well-matched) to the data type of its corresponding column in other result sets.
2. The result sets of all queries must have the same number of columns.
3. Parentheses can be used to alter the sequence of execution.
4. In order to sort the result, an ORDER BY clause should be part of the last select statement. The column names or aliases must be found out by the first select statement, or the positional notation
5. Column names from the first query appear in the result.

Advantage of SET operators in Oracle:

1. Use a set operator to combine multiple queries into a single query
2. These operators are used to combine the information of similar data types from one or more than one table.

Restrictions on the Set Operators:

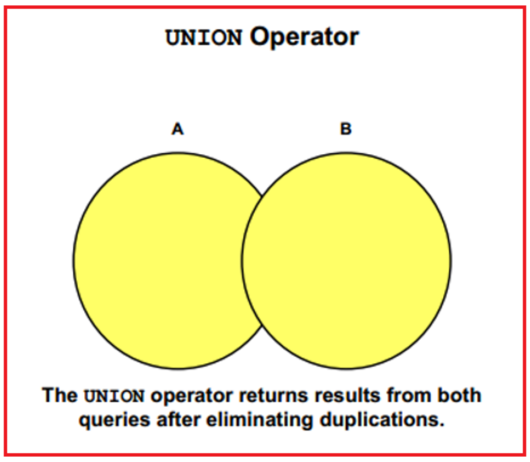
The set operators are subject to the following restrictions:

1. The ORDER BY clause doesn't recognize the column names of the second SELECT
2. The set operators are not valid on columns of type BLOB, CLOB, BFILE, VARRAY, or nested table.
3. The UNION, INTERSECT, and MINUS operators are not valid on LONG columns.
4. Set operations are not allowed on SELECT statements containing TABLE collection expressions.
5. SELECT statements involved in set operations can't use the FOR UPDATE clause.

The generic syntax of a query involving a set operation is:

```
<Component Query>
{UNION | UNION ALL | MINUS | INTERSECT}
<Component Query>
```

UNION Operator in Oracle

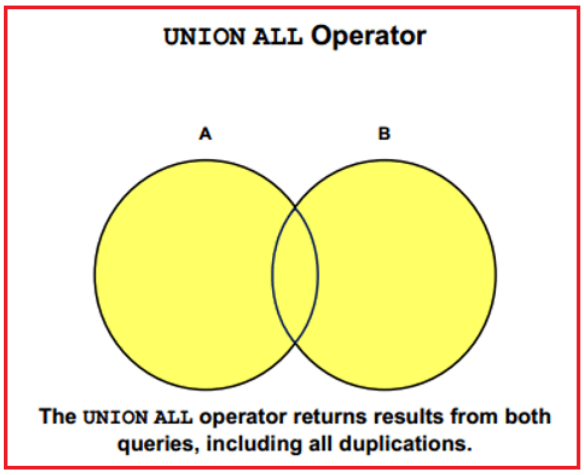

UNION Operator
A
B
The UNION operator returns results from both queries after eliminating duplications.

```
SELECT column_list FROM table1
UNION
SELECT column_list FROM table2;
```

```
SELECT FirstName, LastName, Gender, Department FROM EmployeeUK
UNION
SELECT FirstName, LastName, Gender, Department FROM EmployeeUSA;
```

UNION ALL Operator in Oracle

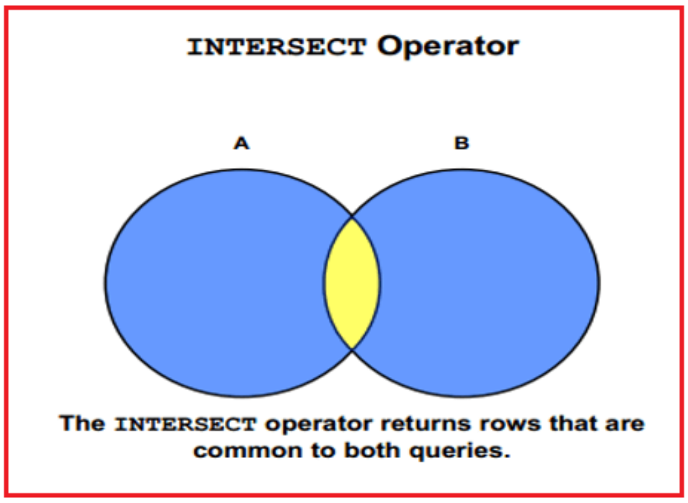
The UNION ALL operator is used to combine the result set of two or more SELECT statements into a single result including the duplicate value.


UNION ALL Operator
A
B
The UNION ALL operator returns results from both queries, including all duplications.

```
SELECT Column_list FROM table1
UNION ALL
SELECT Column_list FROM table2;
```

INTERSECT Operator in Oracle

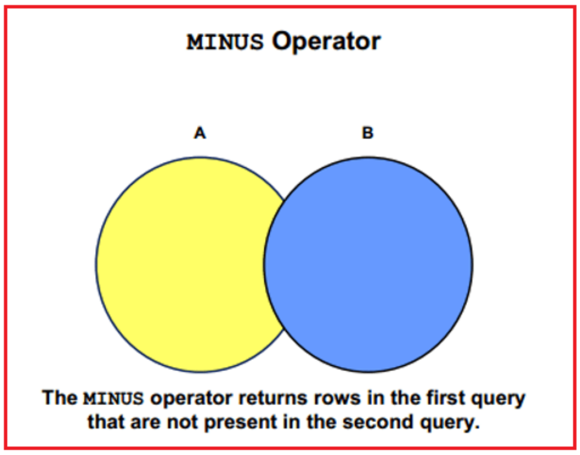
The INTERSECT operator in Oracle is used to combine two result sets and returns the data which are common in both the result set. That means the INTERSECT Operator returns only those rows that are common in both the result sets. Following is the pictorial representation of INTERSECT Operator.


INTERSECT Operator
A
B
The INTERSECT operator returns rows that are common to both queries.

```
SELECT column_lists FROM table_name WHERE condition
INTERSECT
SELECT column_lists FROM table_name WHERE condition;
```

MINUS Operator in Oracle

The MINUS operator in Oracle is used to return unique rows from the left query which isn't present in the right query's results.


MINUS Operator
A
B
The MINUS operator returns rows in the first query that are not present in the second query.

```
SELECT column_lists FROM table_name WHERE condition
MINUS
SELECT column_lists FROM table_name WHERE condition;
```

Is null Operator in Oracle:

The Is Null Operator is used for Comparing nulls in a table. NULL is unknown (or) undefined value in database, NULL != 0 & NULL != space. So, use the" IS " keyword.

Syntax: where <column name> is null;

Example: Write a query to display Employees whose commission is NULL?
SELECT * FROM Employee WHERE COMM IS NULL;

Example: Write a query to display Employees whose commission is NOT NULL?
SELECT * FROM Employee WHERE COMM IS NOT NULL;

Clauses in Oracle

If you want to provide the SQL Query with some additional functionalities such as filtering the records, sorting the records, fetching the records, and grouping the records then you need to use the Clauses along with the SQL Query. **So, in simple words, we can say that clauses are used to provide some additional functionalities.**

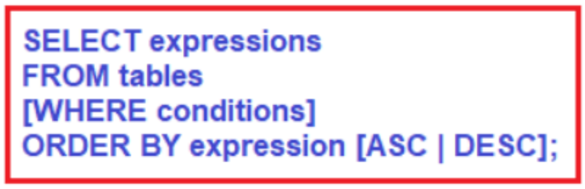
Oracle database supports the following clauses.

- **WHERE**: Filtering rows (before grouping data)
- **ORDER BY**: Sorting values
- **GROUP BY**: Grouping similar data
- **HAVING**: Filtering rows (after grouping data)
- **ROLLUP**: Finding subtotal & grand total (single column)
- **CUBE**: Finding subtotal & grand total (multiple columns)
- **FETCH FIRST:** To Return top n Number of Records

Syntax: **<SQL PER-DEFINE QUERY> + <CLAUSES>;**

ORDER BY Clause

The ORDER BY Clause in Oracle is used for sorting the data either in ascending or descending order based on a specified column or list of columns.



```
SELECT expressions
FROM tables
[WHERE conditions]
ORDER BY expression [ASC | DESC];
```

SELECT * FROM Employee ORDER BY Name;

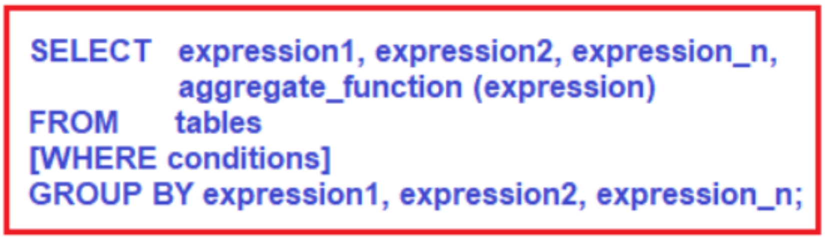
SELECT * FROM Employee ORDER BY Name ASC;

SELECT * FROM Employee ORDER BY Name DESC;

GROUP BY Clause

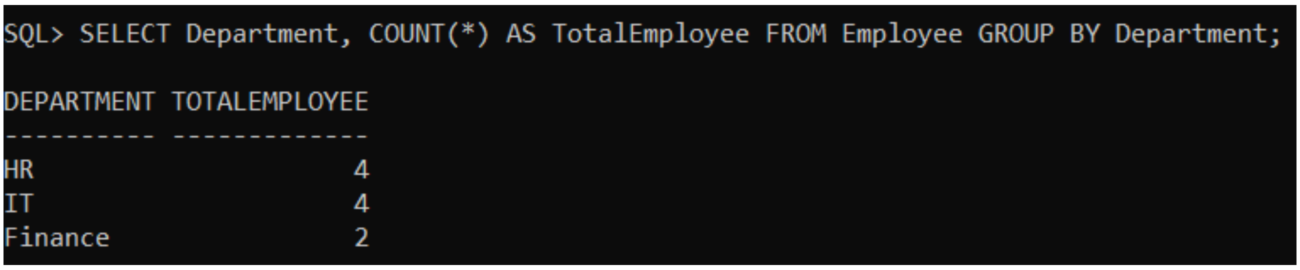
The GROUP BY clause in Oracle is used to group the data together. It aggregates many rows into one. The FROM and WHERE clause creates an intermediate tabular result set and the GROUP BY clause systematically groups the data. The GROUP BY clause can group the result set by one or more columns. That means the Group by Clause divides similar types of records or data as a group and then returns.

If we use group by clause in the query then we should use grouping/aggregate functions such as COUNT(), SUM(), MAX(), MIN(), AVG() functions.



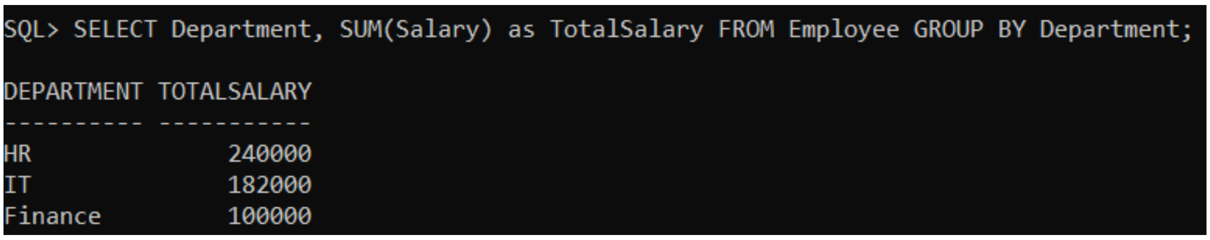
```
SELECT   expression1, expression2, expression_n,
         aggregate_function (expression)
FROM     tables
[WHERE conditions]
GROUP BY expression1, expression2, expression_n;
```

Example: Count the total number of employees in each department from the Employee table.



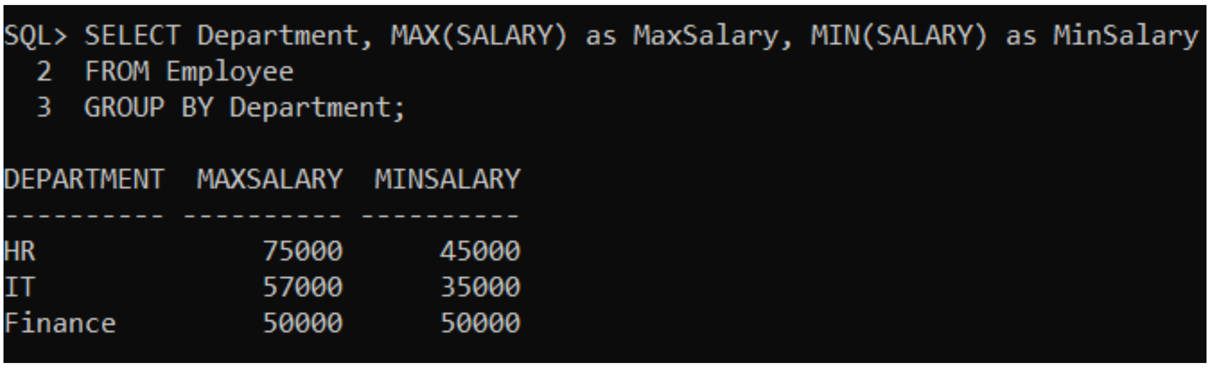
```
SQL> SELECT Department, COUNT(*) AS TotalEmployee FROM Employee GROUP BY Department;

DEPARTMENT TOTALEMPLOYEE
---------- -------------
HR                     4
IT                     4
Finance                2
```

Example: Count the total salary given to each department in the employee table



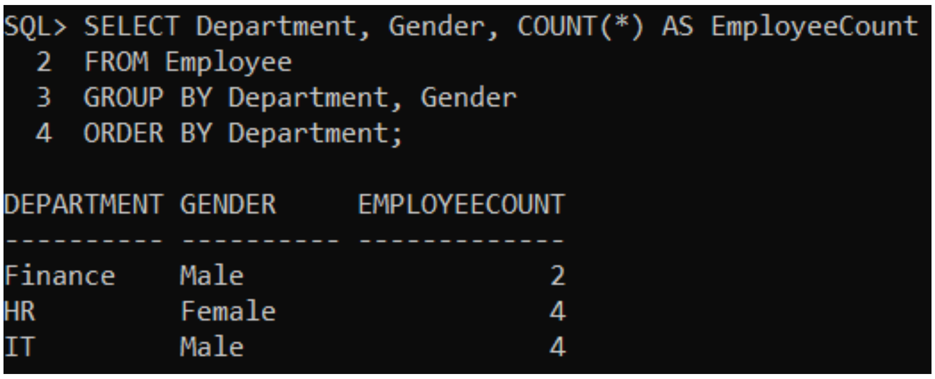
```
SQL> SELECT Department, SUM(Salary) as TotalSalary FROM Employee GROUP BY Department;

DEPARTMENT TOTALSALARY
---------- -----------
HR              240000
IT              182000
Finance         100000
```

Example: Find the Highest and Lowest salary in Each Department in the Employee table



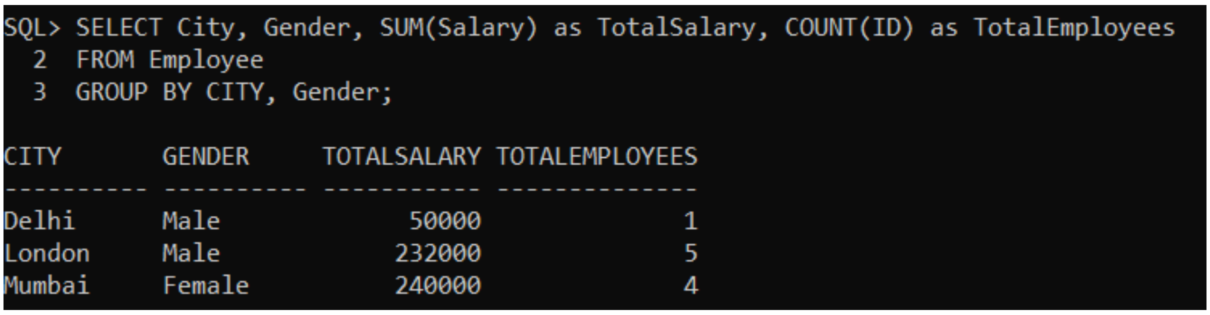
```
SQL> SELECT Department, MAX(SALARY) as MaxSalary, MIN(SALARY) as MinSalary
  2  FROM Employee
  3  GROUP BY Department;

DEPARTMENT  MAXSALARY  MINSALARY
---------- ---------- ----------
HR              75000      45000
IT              57000      35000
Finance         50000      50000
```

Example: Find the number of employees working in each Gender per Department



```
SQL> SELECT Department, Gender, COUNT(*) AS EmployeeCount
  2  FROM Employee
  3  GROUP BY Department, Gender
  4  ORDER BY Department;

DEPARTMENT GENDER     EMPLOYEECOUNT
---------- ---------- -------------
Finance    Male                   2
HR         Female                 4
IT         Male                   4
```

Example: Find the total salaries and the total number of employees by City, and by gender

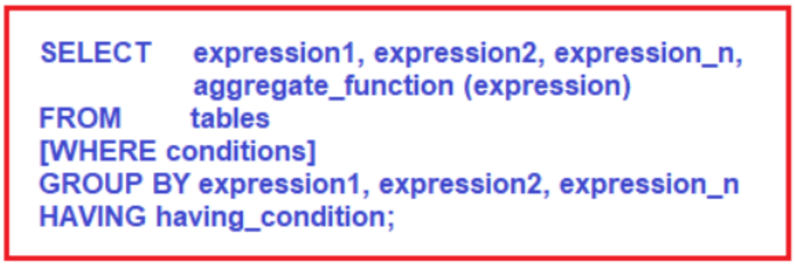


```
SQL> SELECT City, Gender, SUM(Salary) as TotalSalary, COUNT(ID) as TotalEmployees
  2  FROM Employee
  3  GROUP BY CITY, Gender;

CITY       GENDER     TOTALSALARY TOTALEMPLOYEES
---------- ---------- ----------- --------------
Delhi      Male             50000              1
London     Male            232000              5
Mumbai     Female          240000              4
```

Having Clause

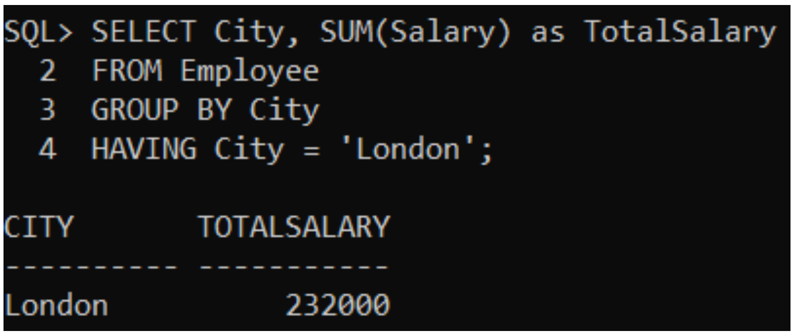
The Having Clause in Oracle is also used for filtering the data just like the where clause. The Having Clause in Oracle is typically used with a GROUP BY clause. That means the Having Clause is used in combination with a GROUP BY clause to restrict the number of groups to be returned by satisfying the condition which is specified using the having clause.

The HAVING clause is the same as the WHERE clause. **The only difference is WHERE clause FILTERS the intermediate data results, while the HAVING clause operates on group rows.** Likewise, WHERE clause, we can use conditions and operators with the HAVING clauses to build complex SQL statements.



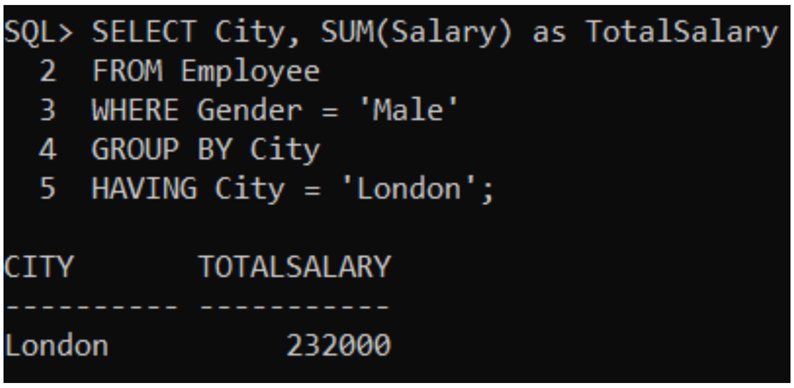
```
SELECT    expression1, expression2, expression_n,
          aggregate_function (expression)
FROM      tables
[WHERE conditions]
GROUP BY expression1, expression2, expression_n
HAVING having_condition;
```

Filtering groups using the HAVING clause in Oracle



```
SQL> SELECT City, SUM(Salary) as TotalSalary
     FROM Employee
     GROUP BY City
     HAVING City = 'London';

CITY       TOTALSALARY
---------- -----------
London          232000
```

Using both Having and Where Clause in Oracle



```
SQL> SELECT City, SUM(Salary) as TotalSalary
     FROM Employee
     WHERE Gender = 'Male'
     GROUP BY City
     HAVING City = 'London';

CITY       TOTALSALARY
---------- -----------
London          232000
```

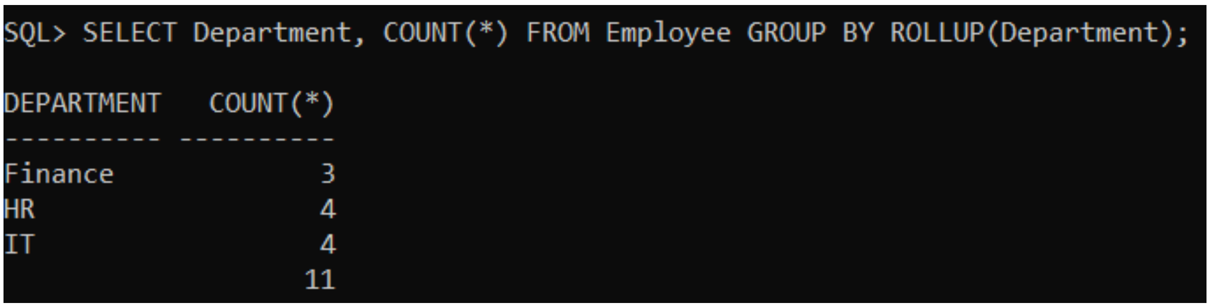
<u>ROLLUP and CUBE Clauses in Oracle</u>

- Special Clauses in Oracle.
- Used for finding subtotal and grand total based on columns.
- Working along with the "GROUP BY" clause.
- ROLLUP will find sub and grand total based on a single column.
- CUBE will find sub and grand totals based on multiple columns.

ROLLUP Syntax in Oracle:

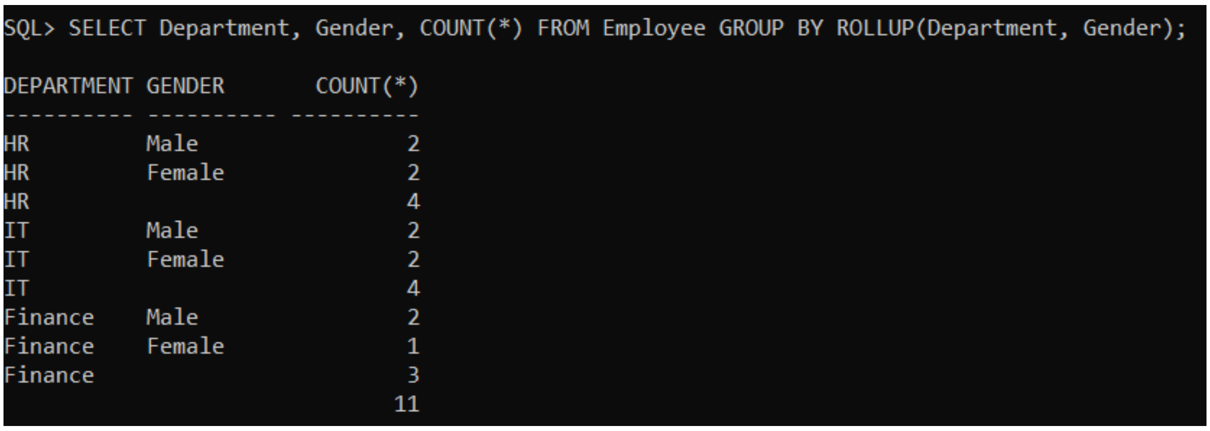
GROUP BY ROLLUP(<COL1>, <COL2>, <COL3>,……..,<COL n>)

Example: ROLLUP with a single column:



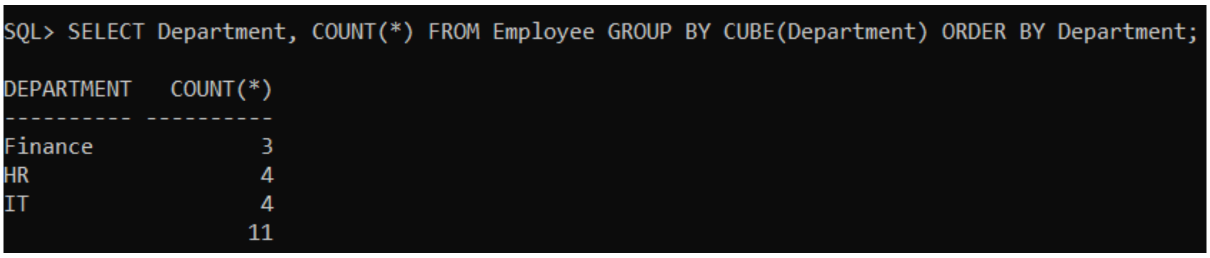
```
SQL> SELECT Department, COUNT(*) FROM Employee GROUP BY ROLLUP(Department);

DEPARTMENT   COUNT(*)
---------- ----------
Finance             3
HR                  4
IT                  4
                   11
```

Example. ROLLUP with multiple columns



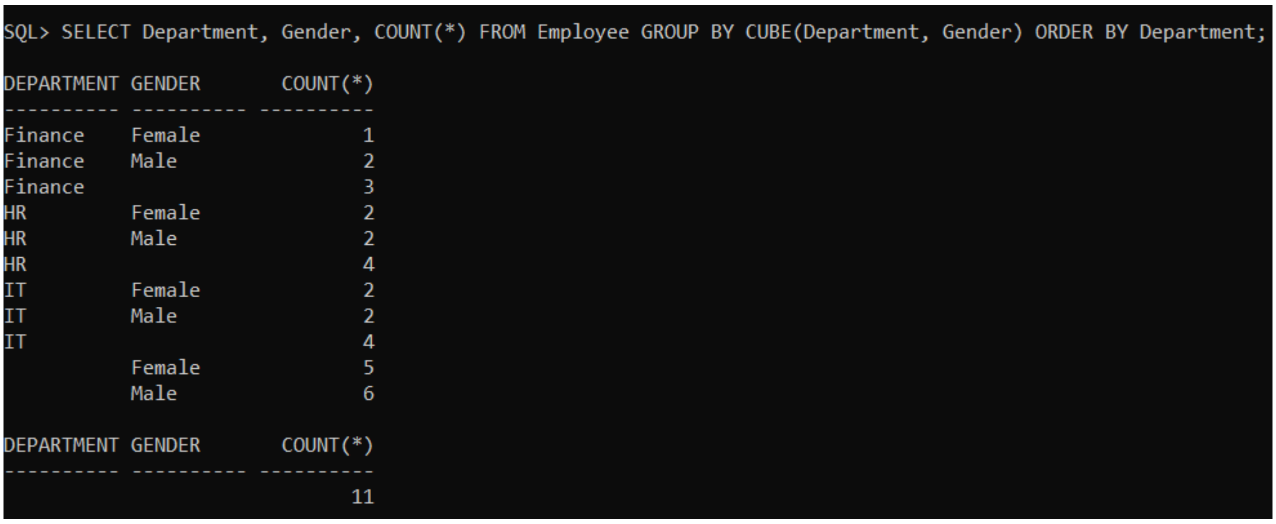
```
SQL> SELECT Department, Gender, COUNT(*) FROM Employee GROUP BY ROLLUP(Department, Gender);

DEPARTMENT GENDER       COUNT(*)
---------- ---------- ----------
HR         Male                2
HR         Female              2
HR                             4
IT         Male                2
IT         Female              2
IT                             4
Finance    Male                2
Finance    Female              1
Finance                        3
                              11
```

Example: CUBE with a single column:



```
SQL> SELECT Department, COUNT(*) FROM Employee GROUP BY CUBE(Department) ORDER BY Department;

DEPARTMENT   COUNT(*)
---------- ----------
Finance             3
HR                  4
IT                  4
                   11
```

Example: CUBE with multiple columns



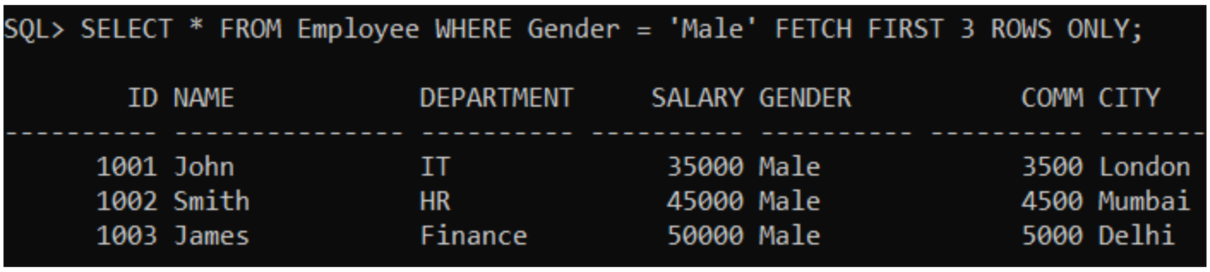
```
SQL> SELECT Department, Gender, COUNT(*) FROM Employee GROUP BY CUBE(Department, Gender) ORDER BY Department;

DEPARTMENT GENDER       COUNT(*)
---------- ---------- ----------
Finance    Female              1
Finance    Male                2
Finance                        3
HR         Female              2
HR         Male                2
HR                             4
IT         Female              2
IT         Male                2
IT                             4
           Female              5
           Male                6

DEPARTMENT GENDER       COUNT(*)
---------- ---------- ----------
                              11
```

<u>FETCH FIRST Clause</u>

The FETCH FIRST Clause in Oracle is used to specify the number of records or rows to return. This FETCH FIRST Clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s)
FETCH FIRST number ROWS ONLY;
```

Example: Fetch the first 3 records from the Employee table where Gender is Male.



```
SQL> SELECT * FROM Employee WHERE Gender = 'Male' FETCH FIRST 3 ROWS ONLY;

        ID NAME            DEPARTMENT     SALARY GENDER           COMM CITY
---------- --------------- ---------- ---------- ---------- ---------- -------
      1001 John            IT              35000 Male             3500 London
      1002 Smith           HR              45000 Male             4500 Mumbai
      1003 James           Finance         50000 Male             5000 Delhi
```

FETCH FIRST PERCENT Clause in Oracle

SELECT * FROM Employee FETCH FIRST 50 PERCENT ROWS ONLY;

Constraints in Oracle

The Oracle Constraints define specific rules to the column(s) data in a database table. While inserting, updating, or deleting the data rows, if the rules of the constraint are not followed, the system will display an error message and the action will be terminated. The Constraints in Oracle are defined while creating a new table. We can also alter the table and add new Constraints. The Constraints are mainly used to maintain data integrity.

Constraints are used to restrict unwanted(invalid) data into the table. All databases are supporting the following constraint types for maintaining data integrity.

1. **NOT NULL Constraint**
2. **UNIQUE KEY Constraint**
3. **CHECK KEY Constraint**
4. **PRIMARY KEY Constraint**
5. **FOREIGN KEY Constraint (REFERENCES Key).**
6. **DEFAULT Constraint**

UNIQUE Constraint in Oracle

UNIQUE Constraint in Oracle is useful to restrict storing of duplicate data row values in a given column or combination of columns. **But it accepts NULL values in that column.**

```
CREATE TABLE Employee (
	Id INT UNIQUE,
	Name VARCHAR(10),
	Email VARCHAR(20) UNIQUE,
);
```

How to add UNIQUE Constraints to existing Columns in Oracle?

ALTER TABLE Employee ADD UNIQUE (Name);

NOT NULL Constraint

By default, the database column stores NULL value means no value in the data row. By defining NOT NULL Constraint to the table column, the default behavior of the database column changes and it does not accept NULL values.

```
CREATE TABLE Student (
      StudentId INT NOT NULL,
      Name VARCHAR2(20) NOT NULL,
      Mobile VARCHAR2(10)
);
```

CHECK Constraint

The Check Constraint in Oracle is used to enforce domain integrity. Domain integrity means the values that are going to store in a table column must be followed by some defined rules such as range, type, and format.

```
CREATE TABLE Employees(
      EmployeeID INT CHECK(EmployeeID BETWEEN 100 AND 1000),
      NAME VARCHAR(20),
      AGE INT NOT NULL CHECK (AGE >= 18),
      DeptID INT CHECK(DeptID > 0 AND DeptID < 100),
      SALARY NUMBER(10)
);
```

Primary Key Constraint

The Primary Key in Oracle is the combination of UNIQUE and NOT NULL Constraint. That means it will not allow either NULL or Duplicate values into a column or columns on which the primary key constraint is applied. Using the primary key, we can enforce **entity integrity** i.e. using the primary key we can uniquely identify each record in a table.

```
CREATE TABLE Employee
(
      Id INT PRIMARY KEY,
      Name VARCHAR2(20),
      Department VARCHAR2(10)
);
```

Composite Primary key in Oracle?

It is also possible in Oracle to create the Primary Key constraint on more than one column and when we do so, it is called a Composite Primary Key.

It is only possible to impose the Composite Primary Key constraint at the table level, it is not possible at the column level.

```
CREATE TABLE EmployeeDetails
(
      Id INT,
      Name VARCHAR2(20),
      Email VARCHAR2(20),
      PRIMARY KEY(Name, Email)
);
```

How to add PRIMARY KEY Constraints to the existing table in Oracle?
ALTER TABLE Test ADD PRIMARY KEY (Id);

How to DELETE PRIMARY Constraints in Oracle?
ALTER TABLE Test DROP PRIMARY KEY;

Foreign Key Constraint

In order to create a link between two tables, we must specify a Foreign Key in one table that references a column in another table. That means the Foreign Key constraint in Oracle is used for binding two tables with each other and then verifying the existence of one table data in other tables.

Note: A foreign key in one TABLE points to either a primary key or a unique key in another table in Oracle. **The foreign key constraints are basically used to enforce referential integrity.**

```
CREATE TABLE Department
(
      Id INT PRIMARY KEY,
      Name VARCHAR2(20),
);


CREATE TABLE Employee
(
      ID INT PRIMARY KEY,
      Name VARCHAR2(20),
      Salary INT,
      DepartmentId INT REFERENCES Department(Id)
);
```

<u>Default Constraint</u>

Default constraints are used to insert a default value to a column when the user doesn't specify a value.

```
CREATE TABLE Employee (
      ID INT NOT NULL,
      Name VARCHAR2(20) NOT NULL,
      Age INT DEFAULT 23,
      Country VARCHAR2(10) DEFAULT 'INDIA'
);
```

Add Default Constraint on existing table:

ALTER TABLE Student MODIFY Country DEFAULT 'INDIA';

How to remove the default value of a column?

ALTER TABLE Student MODIFY Country DEFAULT NULL;

<u>Joins in Oracle</u>

Joins in Oracle are used to retrieve data from multiple tables at a time. Joins in Oracle are again classified into two types are as follows.

NON – ANSI format joins:

When we are retrieving data from multiple tables based on the "WHERE" clause condition then we called a NON-ANSI format join. NON-ANSI joins are not portable. They are again classified into three types are as follows.

1. **Equi Join**
2. **Non-Equi Join**
3. **Self Join**

The syntax for NON-ANSI joins:

SELECT * FROM TABLE NAME1, TABLE NAME2 WHERE <JOIN CONDITION>;

ANSI format joins:

When we are retrieving data from multiple tables with "on" / "using" clause conditions then we called as the join as ANSI format join. ANSI joins are portability (move from one database to another database) They are again classified as follows.

1. **Inner Join**
2. **Outer Join (Left Outer Join, Right Outer Join, and Full Outer Join)**
3. **Cross Join (or) Cartesian Join**
4. **Natural Join**

Syntax for ANSI joins:

SELECT * FROM <TABLE NAME1> <JOIN KEY> <TABLE NAME2 > ON <JOIN CONDITION>;

<u>EQUI Join in Oracle:</u>

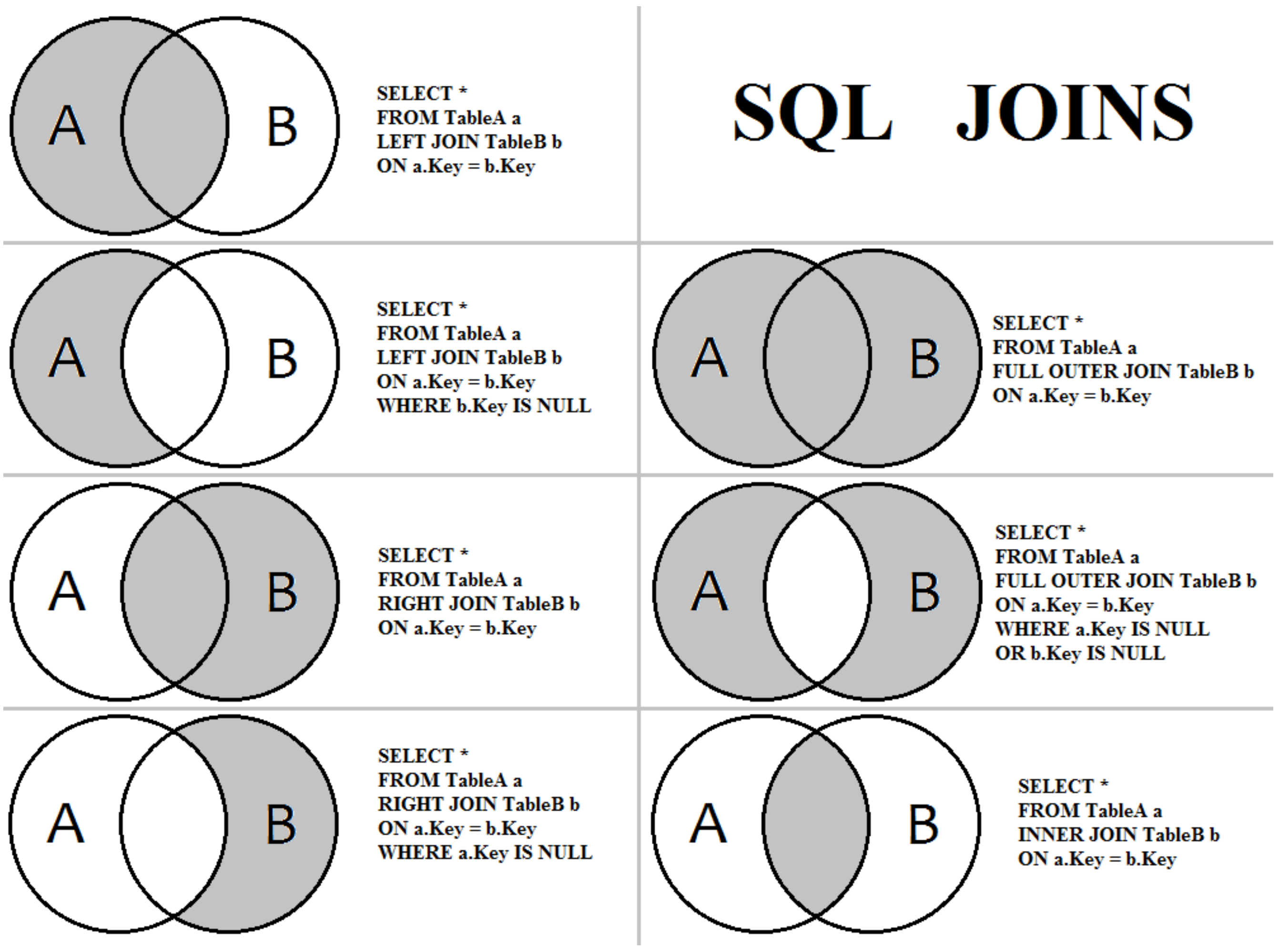
Retrieving data from multiple tables based on “equal operator (=) ” is called an EQUI join. The common column (or) common field datatype must be matched.

Example: Write a Query to retrieve student, course details from tables if CourseId is 20.

```
SELECT * FROM Student S, Course C WHERE S.CourseId = C.CourseId AND
C.CourseId = 20;
```

Write a Query to retrieve student, course details from Course and Student tables by using INNER JOIN.

```
SELECT * FROM Student INNER JOIN Course ON Student.CourseId =
Course.CourseId;
```



```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
```

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
WHERE b.Key IS NULL
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
```

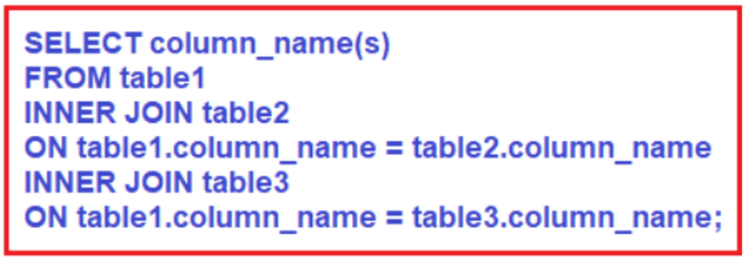
```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
OR b.Key IS NULL
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
```

```
SELECT *
FROM TableA a
INNER JOIN TableB b
ON a.Key = b.Key
```

Joining Three Tables in Oracle:



```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name
INNER JOIN table3
ON table1.column_name = table3.column_name;
```

Fetch the details of such employees who are currently having any projects as well as they must have an address.
The following SQL Query, Inner Joining Employee, Projects, and Address tables to fetch the Employee details, projects, and their addresses.

```
SELECT Employee.EmployeeID, FullName, Department, Gender,
ProjectName, Country, State, City
FROM Employee
INNER JOIN Projects ON Employee.EmployeeID = Projects.EmployeeId
INNER JOIN Address ON Employee.EmployeeID = Address.EmployeeId;
```

Sub Queries in Oracle

In Oracle, a **subquery is a query within a query.** We can create subqueries within our SQL statements. These subqueries can reside in the WHERE clause, in the FROM clause, or in the SELECT clause.

Example: Select records of Employee whose salary is maximum.

SELECT * FROM EMPLOYEE WHERE SAL = (SELECT MAX(SAL) FROM EMPLOYEE);

Single Row Subquery

When a Subquery returns a single value is called a Single Row Subquery. In Single Row Subquery we can use operators such as = , < , > , <= , >=,!=.

Example 1:
Who is getting the first high salary from the Employee table?
SELECT * FROM EMPLOYEE WHERE SAL = (SELECT MAX(SAL) FROM EMPLOYEE);

Example 2:
Whose employee job is the same as the job of 'SMITH'?

```
SELECT * FROM EMPLOYEE WHERE JOB=(SELECT JOB FROM EMPLOYEE WHERE
ENAME='SMITH');
```

Example 3:

Whose salary is more than the max salary of the job is "SALESMAN"?

```
SELECT * FROM EMPLOYEE WHERE SAL>(SELECT MAX(SAL) FROM EMPLOYEE
WHERE JOB='SALESMAN');
```

Example 4

Whose employee job is same as the job of "BLAKE" and who are earning Salary more than "BLAKE" salary?

```
SELECT * FROM EMPLOYEE WHERE JOB=(SELECT JOB FROM EMPLOYEE WHERE
ENAME='BLAKE') AND SAL>(SELECT SAL FROM EMPLOYEE WHERE
ENAME='BLAKE');
```

Example 5

Display employee details who are getting the Second highest Salary in the Employee table?

```
SELECT * FROM EMPLOYEE WHERE SAL=(SELECT MAX(SAL) FROM EMPLOYEE
WHERE SAL< (SELECT MAX(SAL) FROM EMPLOYEE));
```

Subquery with "UPDATE":

To update employee salary with max. salary of Employee table whose empno is 7900?

```
UPDATE EMPLOYEE SET SAL=(SELECT MAX(SAL) FROM EMPLOYEE) WHERE
EMPNO=7900;
```

Subquery with "DELETE":

Delete employee details from the Employee table whose job is the same as the job of 'SCOTT'?

```
DELETE FROM EMPLOYEE WHERE JOB=(SELECT JOB FROM EMPLOYEE WHERE
ENAME='SCOTT');
```

Multiple Row Subquery

When a Subquery returns more than one value is called a Multiple Row Subquery in Oracle. In this Multiple Row Subquery, we can use the operators such as IN, ANY, ALL.

Example1:

Employee details whose employee job is same as the job of the employee "SMITH", and "CLARK"?

```
SELECT * FROM EMPLOYEE WHERE JOB IN(SELECT JOB FROM EMPLOYEE WHERE
ENAME='SMITH' OR ENAME='CLARK');
```

Display all employees, whose salary is more than any "SALESMAN" salary?

```
SELECT * FROM EMPLOYEE WHERE SAL>ANY(SELECT SAL FROM EMPLOYEE
WHERE JOB='SALESMAN');
```

Display all employees, whose salary is more than all "SALESMAN" salaries?

```
SELECT * FROM EMPLOYEE WHERE SAL>ALL(SELECT SAL FROM EMPLOYEE
WHERE JOB='SALESMAN');
```

Multiple Column Subquery

Multiple columns values of the inner query compared with multiple columns values of the outer query is called Multiple Column Subquery in Oracle.
The following is the syntax.

SELECT * FROM <TN> WHERE(<COL1>,<COL2>,............) IN(SELECT <COL1>,<COL2>,....... FROM <TN>);

Example:
Display employee whose JOB, AGE is same as the JOB, AGE of the employee "CLARK"?

SELECT * FROM EMPLOYEE WHERE(JOB, AGE) IN(SELECT JOB, AGE FROM EMPLOYEE WHERE ENAME='CLARK');

DCL (Data Control Language):

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of DCL commands:

- **GRANT**: This command gives users access privileges to the database.
- **REVOKE**: This command withdraws the user's access privileges given by using the GRANT command.

Allow a User to create session

```
GRANT CREATE SESSION TO username;
```

Allow a User to create table

```
GRANT CREATE TABLE TO username;
```

Provide user with space on tablespace to store table

```
ALTER USER username QUOTA UNLIMITED ON SYSTEM;

With fixed memory,
ALTER USER username QUOTA 50m ON SYSTEM;
```

Grant all privilege to a User

```
GRANT sysdba TO username;
```

Note: For granting sysdba permission you must login as sysdba as follows:

connect system as sysdba;

Grant permission to create any table

```
GRANT CREATE ANY TABLE TO username;
```

Grant permission to drop any table

```
GRANT DROP ANY TABLE TO username;
```

Grant multiple permission at a time

```
GRANT CREATE SESSION, CREATE TABLE TO username;
```

To Take Back Permissions

```
REVOKE CREATE TABLE FROM username;

REVOKE CREATE TABLE, CREATE SESSION FROM username;
```