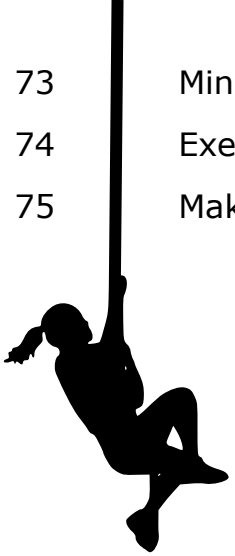
COMPLETE ORACLE SQL

@ Suman Rajak

TABLE OF CONTENTS



What is SQL?

SQl is a language which is followed by every RDBMS. Using this, we can communicate with a database.

CONNECT TO THE DATABASE

```
SQL> connect
     Enter Username:
     Enter Password:
```

Alternative method: -

```
SQL> connect username/password;
```

CLEAR THE SCREEN

```
SQL> clear screen;
```

DATATYPES IN ORACLE SQL

DATA TYPE	DESCRIPTION
NUMBER	40 Digits with decimal and minus sign.
NUMBER(W)	Number of digits specified with decimal and minus sign.
NUMBER(W,D)	Number of digits, number of decimal digits specified with minus sign.
CHAR(W)	Fixed length character data. Maximum size is 2000 (bytes) and default is 1 (byte).
VARCHAR(W)	This variable is pre-reserved and maybe used by Oracle in later future, we should not use it now.
VARCHAR2(W)	String data of specified width. Maximum width is 4000 (bytes).
DATE	Date with default format 26-JAN-14.
LONG	String data of length 0-2 gigabytes. Only one LONG column per table. Can't be used with function, expression or WHERE clause.
RAW(W)	Binary data of specified length. Maximum width is 2000 (bytes).
LONG ROW	Binary data of length 0-2 gigabytes. Only one LONG ROW column per table.
CLOB	A character large object. Maximum size is 4 gigabytes.
BLOB	A binary large object. Maximum size is 4 gigabytes.
BFILE	Contains a locator to a large binary file stored outside the database. Maximum size is 4 gigabytes.

CREATE TABLE

Using this command we can create a table in the database.

Condition:-

- It is a DDL Command
- Table name can be upto 30 char long.
- Table name must begin with alphabet.
- Table name can't contain single or double quote.
- Names are not case sensitive.
- Names can contain a-z , 0-9, _ , $, #
- Names can't be reserved words.

Command: -

```
SQL> Create table tablename (Fieldname datatype (length));
```

Example: -

```
SQL> Create table student (Id number, Name varchar2(10) ,
Address varchar2(10));
```

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------
ID                                                 NUMBER
NAME                                               VARCHAR2(10)
ADRESS                                             VARCHAR2(10)
```

SHOW THE DESCRIPTION OF THE TABLE

```
SQL> desc tablename;
```

INSERT VALUES INTO THE TABLE

```
SQL> Insert into tablename values (values of fields
separated by comma);
```

Alternative method : -

```
SQL> Insert into tablename values (&fields separated by
comma);
```

- Varchar2 needs single quote.
- Number, date does not need single quote

Example: -

```
SQL> Insert into student values (1, ‘Suman’, ‘Purulia’);
```

Alternative Method: -

```
SQL> Insert into student values (&Id, ‘&Name’,
‘&Address’);
Enter the value of Id: -
Enter the value of Name: -
Enter the value of Address: -
```

Insert Specific fields into the table

```
SQL> Insert into tablename (fieldname, fieldname) values
(value, value);
```

Example: -

```
Insert into student2 (Id, Name) values (5, 'Arpan');
```

ID	NAME	ADRESS
1	Suman	Adra
2	Tanu	Moyna
3	Ayan	Rnp
4	HHH	OOO
5	Arpan	

SHOW THE RECORDS OF THE TABLE

Show the full table: -

```
SQL> Select *from tablename;
```

Example: -

```
Select *from Student;
```

```
ID NAME       ADRESS
--- ---------- --------
  1 Suman      Adra
  2 Tanu       Moyna
  3 Ayan       Rnp
  4 HHH        OOO
  5 Arpan
```

Show specific fields of the table: -

```
SQL> Select fieldname, fieldname from tablename;
```

Example: -

```
Select Id, Name from Student;
```

```
ID NAME
---- --------
   1 Suman
   2 Tanu
   3 Ayan
   4 HHH
   5 Arpan
```

DELETE RECORD FROM TABLE

This command will delete records/rows from table with or without condition.

Delete all records: -

```
SQL> Delete from tablename;
```

Delete specific records with a condition: -

```
SQL> Delete from tablename where condition;
```

Example: -

```
SQL> Delete from Student2 where Id=4;
```

```
 ID NAME       ADRESS
--- ---------- -------
  1 Suman      Adra
  2 Tanu       Moyna
  3 Ayan       Rnp
  5 Arpan
```

UPDATE RECORDS IN THE TABLE

Updates records in the table

```
SQL> Update tablename set field= new value where condition;
```

Example: -

```
 ID NAME       ADRESS
--- ---------- -------
  1 Suman      Adra
  2 Tanu       Moyna
  3 Ayan       Rnp
  5 Arpan
```

```
SQL> Update Student2 set name= 'joydip' where name =
'Arpan';
```

```
 ID NAME       ADRESS
--- ---------- -------
  1 Suman      Adra
  2 Tanu       Moyna
  3 Ayan       Rnp
  5 joydip
```

SELECT COMMAND WITH WHERE CONDITION

For all the fields: -

```
SQL> Select *from tablename where condition;
```

For specific fields: -

```
SQL> Select fieldname, fieldname where condition;
```

Concatenate using Select command

```
SQL> Select id || '-' || name || '-' || adress from
student2;
```

Concatenate Statement

Concatenate Operator

```
ID||'-'||NAME||'-'||ADRESS
------------------------------
1-Suman-Adra
2-Tanu-Moyna
3-Ayan-Rnp
5-joydip-
```

Select Null Statement

```
SQL> Select *from tablename where fieldname is null;
```

Example: -

```
SQL> Select *from student2 where adress is null;
```

```
 ID NAME       ADRESS
---- ---------- ----------
  5 joydip
```

Select with Multiple Conditions

```
SQL> Select *from tablename where condition1 and condition2;
```

Example: -

```
SQL> Select *from student2 where id=2 and name = 'Tanu';
```

```
ID NAME       ADRESS
--- ---------- ---------
  2 Tanu       Moyna
```

ORDER BY

It Orders the Output of in ascending or descending order.

For Ascending Order: -

```
SQL> Select *from tablename order by fieldname;
```

For Descending Order: -

```
SQL> Select *from tablename order by fieldname desc;
```

Example: -

```
(1)  SQL> Select *from student2 order by name;
```

```
ID NAME       ADRESS
--- ---------- --------
  3 Ayan       Rnp
  5 Joydip
  1 Suman      Adra
  2 Tanu       Moyna
```

(2) `SQL> Select *from student2 order by id desc;`

```
ID NAME       ADRESS
--- ---------- --------
  5 Joydip
  3 Ayan       Rnp
  2 Tanu       Moyna
  1 Suman      Adra
```

To order using Multiple Fields

If we have some matching records in a field then we have to order using two fields.

```
SQL> Select *from tablename order by field1, field2;
```

If there is some matching field in field1 then we will order using the field2

To display without a specific record

```
SQL> Select *from tablename where not condition;
```

Example: -

```
SQL> Select *from student2 where not id=5;
```

```
ID NAME       ADRESS
--- ---------- --------
  1 Suman      Adra
  2 Tanu       Moyna
  3 Ayan       Rnp
```

LIKE ‘PATTERN’ OPERATOR

Pattern: -

- % :- Represents any value of any length
- _ :- Represents one unknown character
- __ (twice) :- Represents two unknown characters
- To Search using first letter :- *‘ letter %’*
- To Search using middle letter:- *‘%letter%’*
- To search using second letter :- *‘_letter%’*
- To search using last letter :- *‘%letter’*

Example: -

```
 ID NAME       ADRESS
--- ---------- --------
  1 Suman      Adra
  2 Tanu       Moyna
  3 Ayan       Rnp
  5 Joydip
  6 Arpan      Pune
  7 Teen       USA
```

First Letter: -

```
SQL> Select *from student2 where name like 'A%';
```

```
 ID NAME       ADRESS
--- ---------- --------
  3 Ayan       Rnp
  6 Arpan      Pune
```

Last Letter: -

```
SQL> Select *from student2 where name like '%an’;
```

```
 ID NAME       ADRESS
--- ---------- --------
  1 Suman      Adra
  3 Ayan       Rnp
  6 Arpan      Pune
```

Second Letter: -

```
SQL> Select *from student2 where name like '_r%';
```

```
 ID NAME       ADRESS
--- ---------- ---------
  6 Arpan      Pune
```

Middle Letter: -

```
SQL> Select *from student2 where name like '%y%';
```

```
 ID NAME       ADRESS
--- ---------- --------
  3 Ayan       Rnp
  5 Joydip
```

Third Letter: -

```
SQL> Select *from student2 where name like '__m%';
```

```
 ID NAME       ADRESS
--- ---------- ---------
  1 Suman      Adra
```

IN OPERATOR

Using IN Operator we can specify a list of possible values for any column.

Using IN operator: -

```
SQL> Select *from tablename where fieldname in ('fieldname 1', 'fieldname 2');
```

Without using IN operator: -

```
SQL> Select *from tablename where fieldname = 'value' OR fieldname 1 = 'value' OR fieldname 2 = 'value'
```

Example: -

Using IN operator: -

```
SQL> Select *from student2 where adress in ('Rnp','Moyna','Adra');
```

Without using IN operator: -

```
SQl> Select *from student2 where adress = 'Rnp' or adress='Moyna' or adress ='Adra';
```

ID	NAME	ADRESS
1	Suman	Adra
2	Tanu	Moyna
3	Ayan	Rnp

BETWEEN OPERATOR

Using between operator we can select all records which falls in a specific range.

Using BETWEEN operator: -

```
SQL> Select *from tablename where fieldname between
lowerbound AND upperbound;
```

Without using BETWEEN operator: -

```
SQL> Select *from tablename where fieldname >= lowerbound
AND fieldname <= upperbound;
```

Example: -

Using BETWEEN operator: -

```
SQL> Select *from student2 where id between 2 and 5;
```

Without using BETWEEN operator: -

```
SQL> Select *from student2 where id>=2 and id<=5;
```

```
 ID NAME       ADRESS
---- ---------- --------
   2 Tanu       Moyna
   3 Ayan       Rnp
   5 Joydip
```

PSEUDO COLUMNS

A Pseudo Column behaves like a table column, but is not actually stored in the table.

You can select from Pseudo Columns, but you cannot insert, update, or delete their values.

SYSDATE – Current Date and Time

SYSTIMESTAMP – Complete time with millisecond and AM, PM etc.

ROWNUM – sequence number assigned to retrieve rows

ROWID – unique identifier for a row

UID – number associated with a user

USER – UserID of current user

CURRVAL – current value

NEXTVAL – next value

Example: -

SYSDATE: -

```
SQL> select sysdate from dual;
```

```
SYSDATE
---------
10-JUN-22
```

SYSTIMESTAMP: -

```
SQL> select systimestamp from dual;
```

```
SYSTIMESTAMP
-----------------------------------
10-JUN-22 01.33.19.474000 AM +05:30
```

UID: -

```
SQL> select uid from dual;
```

```
  UID
------
     5
```

UID: -

```
SQL> select user from dual;
```

```
USER
--------
SYSTEM
```

Currval and Nextval

```
SQL> CREATE SEQUENCE id_seq
     MINVALUE 1
     MAXVALUE 9999
     START WITH 1
     INCREMENT BY 1
     CACHE 20
```

With respect to a sequence, the cache option specifies how many sequence values will be stored in memory for faster access.

'Nocache' means that none of the sequence values are stored i memory.

Example: -

1. First, we have to create a sequence.

```
SQL> CREATE SEQUENCE id_seq
  2 MINVALUE 1
  3 MAXVALUE 9999
  4 START WITH 1
  5 INCREMENT BY 1
  6  CACHE 20;
```

2. Now we will insert the value of ID in a table using this sequence.

```
SQL> Insert into tablename values (fieldname
sequencename.nextval);
```

Example: -

```
SQL> create table tb1 (id number, name varchar2(20));
```

```
SQL> insert into tb1 vlaues (id_seq.nextval , 'ram');
SQL> insert into tb1 values (id_seq.nextval , 'rahim');
SQL> insert into tb1 values (id_seq.nextval , 'shyam');
SQL> insert into tb1 values (id_seq.nextval , 'bharat');
```

```
ID NAME
--- --------
  1 ram
  2 rahim
  3 shyam
  4 bharat
```

```
SQL> insert into tb1 values (id_seq.currval , 'sundar');
```

```
ID NAME
--- --------
  1 ram
  2 rahim
  3 shyam
  4 bharat
  4 sundar
```

ROWID AND ROWNUM

ROWID is a unique pseudo number assigned to each row uniquely, they never collapse to match different table.

ROWNUM returns the number of rows for a resultant query.

Example: -

```
SQL> select rowid from tb1;
```

```
ROWID
------------------
AAADgqAABAAAKjaAAA
AAADgqAABAAAKjaAAB
AAADgqAABAAAKjaAAC
AAADgqAABAAAKjaAAD
AAADgqAABAAAKjaAAE
```

```
SQL> select rownum from tb1;
```

```
 ROWNUM
-------
      1
      2
      3
      4
      5
```

Q) Select less than 3rd row without any ID field.

Ans:-

```
SQL> select *from tb1 where rownum<3;
```

```
 ID NAME
--- --------
  1 ram
  2 rahim
```

RENAME TABLE

```
SQL> Rename oldtablename to newtablename;
```

DROP TABLE

```
SQL> Drop table tablename;
```

ROLLBACK

To use rollback we have to make a savepoint. Then we will change something in the database and will rollback to the savepoint. We will observe that the changes will be removed due to the rollback.

We can undo the insert , update and delete command using rollback;

STEP 1 → `SQL> Savepoint savepoint_name;`

STEP 2 → NOW MAKE SOME CHANGES IN THE DATABASE.

STEP 3 → `SQL> Rollback to savepoint_name;`

STEP 4 → THE CHANGES WILL BE REMOVED.

Example: -

```
SQL> savepoint s;

Savepoint created.

SQL> create table t1 (id number , name varchar2(50));

Table created.

SQL> insert into t1 values(1, 'Ram');

1 row created.

SQL> insert into t1 values(2,'Shyam');

1 row created.

SQL> select *from t1;

        ID NAME
---------- --------------------------------------------
         1 Ram
         2 Shyam
```

```
SQL> roll back to s;
Rollback complete.
SQL> select *from t1;

no rows selected
```

COMMIT

Commit is used to permanently save in the database. We can permanently save the insert , update and delete command using rollback;

```
SQL> COMMIT;
```

Example: -

```
SQL> savepoint s9;

Savepoint created.

SQL> create table t2 (id number , name varchar2(50));

Table created.

SQL> insert into t2 values(1, 'Ram');

1 row created.

SQL> insert into t2 values(2, 'Shyam');

1 row created.

SQL> select *from t2;

        ID NAME
---------- ----------------------------------------
         1 Ram
         2 Shyam

SQL> commit;

Commit complete.

SQL> roll back to s9;
Rollback complete.
SQL> select *from t2;

        ID NAME
---------- ----------------------------------------
         1 Ram
         2 Shyam
```

CREATING TABLESPACE

```
SQL>CONNECT username/password;
SQL>CREATE TABLESPACE TablespaceName DATAFILE
   2 'C://oraclexe/oradata/xe/TablespaceName.dbf'
   3 SIZE 50M;
```

Will create a tablespace
Of size 50 MB

```
SQL> create tablespace DEMO DATAFILE
  2  'C:\oraclexe\oradata\XE\DEMO.DBF'
  3  SIZE 50M;

Tablespace created.
```

CREATING USER

```
SQL>CREATE USER UserName IDENTIFIED BY Password
   DEFAULT TABLESPACE TablespaceName
   TEMPORARY TABLESPACE Temp
   QUOTA UNLIMITED ON TablespaceName;
```

CREATING ROLE

```
SQL>CREATE ROLE RoleName
```

ASSIGNING PERMISSION ON ROLE

```
SQL>GRANT (roles) CREATE TABLE, CREATE SESSION TO RoleName;
```

PASSING ROLE TO USER

```
SQL>GRANT RoleName TO UserName;
```

Example: -

```
SQL> CREATE USER SUMAN IDENTIFIED BY SUMANRAJAK
  2   DEFAULT TABLESPACE DEMO
  3  TEMPORARY TABLESPACE TEMP
  4  QUOTA UNLIMITED ON DEMO;

User created.

SQL> CREATE ROLE GM;

Role created.

SQL> GRANT CREATE TABLE, CREATE SESSION TO GM;

Grant succeeded.

SQL> GRANT GM TO SUMAN;

Grant succeeded.

SQL> CONNECT
Enter user-name: SUMAN
Enter password:
Connected.
```

Now the user can create tables and can log in to the database because of the "create session" role.

ALTER TABLESPACE

If a user has "alter tablespace" privilege then only he can use the alter tablespace command.

A tablespace can have multiple data files.

Add New Datafile to an Existing Tablespace

```
SQL> alter tablespace tablespacename add datafile
'C:\oraclexe\oradata\XE\datafilename.DBF'
2    SIZE size in mb;
```

See Total Datafiles List

```
SQL> select file_name from dba_data_files;
```

```
FILE_NAME
------------------------------------
C:\ORACLEXE\ORADATA\XE\USERS.DBF
C:\ORACLEXE\ORADATA\XE\SYSAUX.DBF
C:\ORACLEXE\ORADATA\XE\UNDO.DBF
C:\ORACLEXE\ORADATA\XE\SYSTEM.DBF
C:\ORACLEXE\ORADATA\XE\DEMO.DBF
C:\ORACLEXE\ORADATA\XE\A.DBF
C:\ORACLEXE\ORADATA\XE\B.DBF
```

See Total tablespace list

```
SQL> select tablespace_name from dba_data_files;
```

```
TABLESPACE_NAME
--------------------
USERS
SYSAUX
UNDO
SYSTEM
DEMO
ABC
ABC
```

Remove Datafile from a tablespace

We can only remove a datafile from a tablespace when it have multiple datafiles in it. Because it is mandatory for a tablespace to have at least one datafile in it.

```
SQL> alter tablespace tablespacename drop datafile
'C:\oraclexe\oradata\XE\datafilename.DBF';
```

DROP TABLESPACE

```
SQL> drop tablespace tablespacename including contents and
datafiles;
```

INSERT MULTUIPLE RECORDS IN ONE OR MORE TABLE AT ONCE

```
SQL> insert all
  2 into tablename values ()
  3 into tablename values ()
  4 into tablename values ()
  5 select *from dual;
```

Example: -

(1) We are creating a table named tab1 and inserting 3 records at once.

```
SQL> insert all
 2 into tab1 values (1, ‘ram’)
 3 into tab1values (2,’shyam’)
 4 into tab1 values (3,’laxman’)
 5 select *from dual;
```

```
SQL> select *from tab1;

        ID NAME
---------- --------------
         1 ram
         2 shyam
         3 laxman
```

(2) we are creating another table named tab2 and inserting multiple records in both table at once.

```
SQL> insert all
 2 into tab1 values (4, 'abc')
 3 into tab1 values (5, 'xyz')
 4 into tab2 values (1, 'tyu')
 5 into tab2 values (2, 'hjk')
 6 select *from dual;
```

```
SQL> select *from tab1;

        ID NAME
---------- --------------
         1 ram
         2 shyam
         3 laxman
         4 abc
         5 xyz
```

```
SQL> select *from tab2;

        ID NAME
---------- --------------
         1 tyu
         2 hjk
```

GRANT A USER TABLE TO ANOTHER USER

First Make 2 users

Make 2 roles

Assign them the roles

Now connect to first user (let, 'sr') and create a table 'te1'

Generally, we cannot see the table created by first user by connecting the second user (let, 'tanu')

Now we will grant tanu to 'view' the table of sr

```
SQL> connect sr/sr;
SQL> grant select on te1 to tanu;
```

'Select' is a permission to view only. We can also use other permissions like 'insert', 'update', 'delete'.

Now Connect to tanu and view the table of sr by using this query

```
SQL> select *from sr.te1;
```

Revoke the permission from tanu : -

```
SQL> revoke select on te1 from tanu;
```

DROP ROLES

Connect to administrative account i.e., using system username and use this query

```
SQL> Drop Role rolename cascade.
```

To delete the associated users which are associated with the role

DROP USER

Connect to administrative account i.e., using system username and use this query

```
SQL> Drop user username cascade;
```

REFERENTIAL INTEGRITY OR FOREIN KEY

Parent Table: -

```
SQL> create table student_add (roll_no number primary key, name
varchar2(50), city varchar2(10), mobile number (13), pin number (6));
```

Child Table: -

```
SQL> create table student_marks (roll_no number references student_add
on delete cascade, subject varchar2(50), marks number (3));
```

Because of 'on delete cascade' if we delete a record from the parent table then it will be automatically deleted from the child table.

COLUMN ALIAS

Using this we can change the column head name in the output.

```
SQL> select fieldname "desired field name in output" from tablename;
```

Example: -

```
 ID NAME       ADRESS
--- ---------- --------
  1 Suman      Adra
  2 Tanu       Moyna
  3 Ayan       Rnp
  5 Joydip
  6 Arpan      Pune
  7 Teen       USA
```

```
SQL> select adress "Home", name from student2;
```

```
Home       NAME
---------- --------
Adra       Suman
Moyna      Tanu
Rnp        Ayan
           Joydip
Pune       Arpan
USA        Teen
```

```
SQL> select id+10 "New ID", name, adress "HOME" from
student2;
```

In this way, we can do calculations inside the query.

```
 New ID NAME       HOME
------- ---------- ----------
     11 Suman      Adra
     12 Tanu       Moyna
     13 Ayan       Rnp
     15 Joydip
     16 Arpan      Pune
     17 Teen       USA
```

EQUI JOIN

In this join we use the equal operator.

```
SQL> Select fieldname, fieldname from table1, table2 where
table1.field = table2.field;
```

Example: -

```
  DNO DNAME
----- ---------
   10 MCA
   20 MBA
   30 BCA
   40 MTECH
```

DEPT

```
 ENO ENAME                     DNO
----- -------------------- ----------
    1 Suman                       10
    2 Tanu                        20
    3 Ayan                        10
    4 Joy                         50
```

EMPL

Between these two tables we will fetch the matching fields.

```
SQL> Select eno,ename ,dname from empl,dept where dept.dno=empl.dno;
```

```
 ENO ENAME                DNAME
----- -------------------- ----------
    3 Ayan                 MCA
    1 Suman                MCA
    2 Tanu                 MBA
```

NON EQUI JOIN

Here we use other operators like >, <, > =, < =, < > (Not Equals).

```
SQL> Select fieldname, fieldname from table1, table2 where
table1.field operator table2.field;
```

Example: -

(1)

```
  DNO DNAME
----- ---------
   10 MCA
   20 MBA
   30 BCA
   40 MTECH
```

DEPT

```
 ENO ENAME                     DNO
----- -------------------- ----------
    1 Suman                       10
    2 Tanu                        20
    3 Ayan                        10
    4 Joy                         50
```

EMPL

We will fetch the dno of the dept table which are greater than the dno in the empl table

```
SQL> Select eno,ename ,dname from empl,dept where
dept.dno>empl.dno;
```

```
 ENO ENAME                DNAME
----- -------------------- ---------
    1 Suman                MBA
    1 Suman                BCA
    1 Suman                MTECH
    2 Tanu                 BCA
    2 Tanu                 MTECH
    3 Ayan                 MBA
    3 Ayan                 BCA
    3 Ayan                 MTECH
```

In dept table,
10 is greater than no one in the empl table.
20 is greater than 10 and 10 in the empl table
30 is greater than 10,20,10 in the empl table
40 is greater than 10,20,10 in the empl table

So we will have total 0+2+3+3 = 8 records.

(2)

DNO	DNAME
10	MCA
20	MBA
30	BCA
40	MTECH

DEPT

ENO	ENAME	DNO
1	Suman	10
2	Tanu	20
3	Ayan	10
4	Joy	50

EMPL

We will fetch the dno of the dept table which are lesser than the dno in the empl table

```
SQL> Select eno,ename ,dname from empl,dept where
dept.dno>empl.dno;
```

ENO	ENAME	DNAME
1	Suman	MBA
1	Suman	BCA
1	Suman	MTECH
2	Tanu	BCA
2	Tanu	MTECH
3	Ayan	MBA
3	Ayan	BCA
3	Ayan	MTECH

In dept table,
10 lesser than 20,50 in the empl table.
20 is lesser than 50 in the empl table
30 is lesser than 50 in the empl table
40 is lesser than 50 in the empl table

So we will have total 2+1+1+1 = 5 records.

TABLE ALIAS

```
SQL> Select fieldname, fieldname from table1 table1_new_name, table2
table2_new_name where condition
```

Example: -

```
SQL> select eno,ename , dname from empl e , dept d where e.dno=d.dno;
```

TABLE ALIAS i.e.; Table Temporary New Names

SELF JOIN

In this join a table joins with itself with two different temporary names using table alias.

```
SQL> Select tablenamealias.fieldname , tablenamealias.fieldname from
tablename tablenamealias1 , tablename tablenamealias2 where
tablenamealias1.fieldname condition tablenamealias2.fieldname.
```

Example: -

ENO	ENAME	DNO
1	Suman	10
2	Tanu	20
3	Ayan	10
4	Joy	50

EMPL

```
SQL> select e1.eno, e2.ename, e1.dno from empl e1 , empl e2
where e1.eno = e2.eno;
```

ENO	ENAME	DNO
1	Suman	10
2	Tanu	20
3	Ayan	10
4	Joy	50

```
SQL> select e1.eno, e2.ename, e1.dno from empl e1 , empl e2
where e1.eno > e2.eno;
```

ENO	ENAME	DNO
2	Suman	20
3	Suman	10
3	Tanu	10
4	Suman	50
4	Tanu	50
4	Ayan	50

```
SQL> select e1.eno, e2.ename, e1.dno from empl e1 , empl e2
where e1.eno < e2.eno;
```

ENO	ENAME	DNO
1	Tanu	10
1	Ayan	10
1	Joy	10
2	Ayan	20
2	Joy	20
3	Joy	10

NATURAL JOIN

Natural join returns the matching column in both of the tables.

```
SQL> Select fieldname, fieldname, fieldname from table1
natural join table2;
```

Example: -

DNO	DNAME
10	MCA
20	MBA
30	BCA
40	MTECH

DEPT

ENO	ENAME	DNO
1	Suman	10
2	Tanu	20
3	Ayan	10
4	Joy	50

EMPL

```
SQL> select eno, ename, dno, dname from empl natural join
dept;
```

ENO	ENAME	DNO	DNAME
3	Ayan	10	MCA
1	Suman	10	MCA
2	Tanu	20	MBA

CROSS JOIN

In the output the rows are merged of both of the tables without any condition

```
SQL> Select *from table1, table2.
```

OR,

```
SQL> Select *from table1 cross join table2.
```

Example: -

DNO	DNAME
10	MCA
20	MBA
30	BCA
40	MTECH

DEPT

ENO	ENAME	DNO
1	Suman	10
2	Tanu	20
3	Ayan	10
4	Joy	50

EMPL

```
SQL> select *from empl, dept;
```

OR,

```
SQL> select *from empl cross join dept;
```

ENO	ENAME	DNO	DNO	DNAME
1	Suman	10	10	MCA
1	Suman	10	20	MBA
1	Suman	10	30	BCA
1	Suman	10	40	MTECH
2	Tanu	20	10	MCA
2	Tanu	20	20	MBA
2	Tanu	20	30	BCA
2	Tanu	20	40	MTECH
3	Ayan	10	10	MCA
3	Ayan	10	20	MBA
3	Ayan	10	30	BCA
3	Ayan	10	40	MTECH
4	Joy	50	10	MCA
4	Joy	50	20	MBA
4	Joy	50	30	BCA
4	Joy	50	40	MTECH

LEFT OUTER JOIN

It returns all the tuples of the left relation and only the matching tuples from the right relation.

```
SQL> Select *from left_table, right_table where
left_table.matching_field = right_table.matching_field (+)
```

OR,

```
SQL> Select *from left_table left outer join right_table  on
left_table.matching_field = right_table.matching_field
```

Example: -

ENO	ENAME	DNO
1	Suman	10
2	Tanu	20
3	Ayan	10
4	Joy	50

EMPL

DNO	DNAME
10	MCA
20	MBA
30	BCA
40	MTECH

DEPT

```
SQL> select *from empl,dept where empl.dno=dept.dno (+);
```

OR,

```
Select *from empl left outer join dept on empl.dno=dept.dno;
```

ENO	ENAME	DNO	DNO	DNAME
3	Ayan	10	10	MCA
1	Suman	10	10	MCA
2	Tanu	20	20	MBA
4	Joy	50		

RIGHT OUTER JOIN

It returns all the tuples of the right relation and only the matching tuples from the left relation.

```
SQL> Select *from left_table, right_table where
left_table.matching_field (+) = right_table.matching_field
```

OR,

```
SQL> Select *from left_table right outer join right_table
on left_table.matching_field = right_table.matching_field
```

Example: -

```
  ENO ENAME                       DNO
----- -------------------- ----------
    1 Suman                        10
    2 Tanu                         20
    3 Ayan                         10
    4 Joy                          50
```

EMPL

```
  DNO DNAME
------ ---------
   10 MCA
   20 MBA
   30 BCA
   40 MTECH
```

DEPT

```
SQL> select *from empl,dept where empl.dno(+)=dept.dno;
```

OR,

```
Select *from empl right outer join dept on
empl.dno=dept.dno;
```

```
ENO ENAME                       DNO        DNO DNAME
---- -------------------- ---------- ---------- -----------
   1 Suman                        10         10 MCA
   2 Tanu                         20         20 MBA
   3 Ayan                         10         10 MCA
                                             30 BCA
                                             40 MTECH
```

FULL OUTER JOIN

It returns all the tuples from both of the relations even if they are not matching.

```
SQL> Select *from left_table full outer join right_table  on
left_table.matching_field = right_table.matching_field
```

Example: -

```
 ENO ENAME                       DNO
----- -------------------- ----------
    1 Suman                        10
    2 Tanu                         20
    3 Ayan                         10
    4 Joy                          50
```

EMPL

```
  DNO DNAME
------ ---------
    10 MCA
    20 MBA
    30 BCA
    40 MTECH
```

DEPT

```
SQL> select *from empl full outer join dept on
empl.dno=dept.dno;
```

```
ENO ENAME                       DNO        DNO DNAME
---- -------------------- ---------- ---------- ---------
   3 Ayan                         10         10 MCA
   1 Suman                        10         10 MCA
   2 Tanu                         20         20 MBA
   4 Joy                          50
                                             30 BCA
                                             40 MTECH
```

ANTI JOIN OR NOT IN OPERATOR

It returns all the records of the first table with are not matching with the second table.

```
SQL> Select *from table1 where matching_field not in (Select
*from table2 where condition);
```

Example: -

```
 ENO ENAME                          DNO
----- -------------------- ----------
    1 Suman                         10
    2 Tanu                          20
    3 Ayan                          10
    4 Joy                           50
```

EMPL

```
  DNO DNAME
------ ---------
   10 MCA
   20 MBA
   30 BCA
   40 MTECH
```

DEPT

```
SQL> select *from empl where dno not in  (select dno from
dept where dname='MBA');
```

```
 ENO ENAME                          DNO
---- -------------------- ----------
   1 Suman                         10
   3 Ayan                          10
   4 Joy                           50
```

COLUMN FORMAT

We can temporarily format any column as per our requirement.

```
SQL> Column column_name format format_statement;
```

Example: -

```
  ENO ENAME                          DNO     SALARY
----- -------------------- ---------- ----------
    1 Suman                          10      70000
    2 Tanu                           20      80000
    3 Ayan                           10      80000
    4 Joy                            50      75000
```

```
SQL> column salary format 9,99,999.99;
```

The values of the Salary column will be separated by a comma.

```
SQL> select *from empl;

       ENO ENAME                          DNO       SALARY
---------- -------------------- ---------- ------------
         1 Suman                          10    70,000.00
         2 Tanu                           20    80,000.00
         3 Ayan                           10    80,000.00
         4 Joy                            50    75,000.00
```

```
SQL> column ename format a3 trunc;
```

It will show only the first three letters of the values of the ename column

```
SQL> select *from empl;

       ENO ENA        DNO       SALARY
---------- --- ---------- ------------
         1 Sum         10    70,000.00
         2 Tan         20    80,000.00
         3 Aya         10    80,000.00
         4 Joy         50    75,000.00
```

CLEAR COLUMN FORMATTING

```
SQL> clear column;
```

OR,

Restart the 'Run SQL Command Line'

AGREEGATE FUNCTIONS IN ORACLE SQL

AVG(x)	It returns average of any numeric column. SELECT AVG(SALARY) FROM EMPLOYEE;
COUNT(x)	It returns total number records in any column. SELECT COUNT(SALARY) FROM EMPLOYEE; SELECT COUNT(NAME) FROM EMPLOYEE;
COUNT(*)	It returns total number of records in the table. SELECT COUNT(*) FROM EMPLOYEE;
COUNT(DISTINCT x)	It returns total number of distinct records in any column. SELECT COUNT(DISTINCT name) FROM EMPLOYEE;
MAX(x)	It returns maximum value in any column. SELECT MAX(SALARY) FROM EMPLOYEE;
MAX(DISTINCT x)	It returns maximum distinct value in any column.
MIN(x)	It returns minimum value in any column. SELECT MIN(SALARY) FROM EMPLOYEE;
MIN(DISTINCT x)	It returns minimum distinct value in any column.
SUM(x)	It returns total sum of any column. SELECT SUM(SALARY) FROM EMPLOYEE;
SUM(DISTINCT x)	It returns total sum of distinct values of any column.
STDDEV(x)	It returns standard deviation of any column. SELECT STDDEV(SALARY) FROM EMPLOYEE;
VARIANCE(x)	It returns variance of any column. SELECT VARIANCE(SALARY) FROM EMPLOYEE;

Example: -

```
 ENO ENAME                     DNO     SALARY
---- -------------------- ---------- ----------
   1 Suman                        10      70000
   2 Tanu                         20      80000
   3 Ayan                         10      80000
   4 Joy                          50      75000
```

(1) AVG

```
SQL> Select avg (salary) from empl;
```

```
salary average
--------------
         76250
```

(2) COUNT (X)

```
SQL> select count(salary) from empl;
```

```
COUNT(SALARY)
-------------
            4
```

(3) COUNT (*)

```
SQL> select count(*) from empl;
```

```
  COUNT(*)
----------
         4
```

(4) COUNT (DISTINCT X)

```
SQL> select count (distinct salary) from empl;
```

```
COUNT(DISTINCTSALARY)
---------------------
                    3
```

(5) MAX(X)

```
SQL> select max(salary) from empl;
```

```
MAX(SALARY)
-----------
      80000
```

(6) MIN (X)

```
SQL> select min(salary) from empl;
```

```
MIN(SALARY)
-----------
      70000
```

(7) SUM (X)

```
SQL> select sum(salary) from empl;
```

```
SUM(SALARY)
-----------
     305000
```

(8) STDDEV (X)

```
SQL> select stddev(salary) from empl;
```

```
STDDEV(SALARY)
--------------
    4787.13554
```

(9) VARIANCE (X)

```
SQL> select variance (salary) from empl;
```

```
VARIANCE(SALARY)
----------------
      22916666.7
```

CHARACTER FUNCTIONS IN ORACLE

INITCAP(data)	It converts first letter of string to uppercase. SELECT INITCAP('ABHIMANYU') FROM DUAL; Output: Abhimanyu
LENGTH(data)	It returns the length of the string. SELECT LENGTH('i am abhimanyu') FROM DUAL; Output: 14. Here White Spaces are also counted
SUBSTR(data,x,y)	It returns substring from string. SELECT SUBSTR('i am abhimanyu',3,2) FROM DUAL; Output: am
INSTR(data,x)	It will find location of string 'x' in 'data' string. SELECT INSTR('i am abhimanyu','am') FROM DUAL; Output: 3
GREATEST(n1,n2,n3,..)	It returns greatest number from group of numbers. SELECT GREATEST(56,34,2,45,57) FROM DUAL; Output: 57
LEAST(n1,n2,n3,..)	It returns smallest number from group of numbers. SELECT GREATEST(56,34,2,45,57) FROM DUAL; Output: 2

It will start from 3 and Will take 2 words.

Example: -

INITCAP

```
SQL> select initcap ('suman') from dual;
```

```
INITC
-----
Suman
```

LENGTH

```
SQL> select length ('i am suman') from dual;
```

```
LENGTH('IAMSUMAN')
------------------
                10
```

SUBSTR

```
SQL> select substr ('i am suman',3,2) from dual;
```

```
SU
--
am
```

INSTR

```
SQL> select instr('i am suman','am') from dual;
```

```
INSTR('IAMSUMAN','AM')
----------------------
                     3
```

GREATEST

```
SQL> select greatest(56,78,45,98,67) from dual;
```

```
GREATEST(56,78,45,98,67)
------------------------
                      98
```

LEAST

```
SQL> select least(56,78,45,98,67) from dual;
```

```
LEAST(56,78,45,98,67)
---------------------
                   45
```

NUMBER FUNCTIONS IN ORACLE

ABS(data)	It returns absolute value i.e. positive value. SELECT ABS(-345) FROM DUAL; Output: 345
CEIL(data)	It returns smallest integer greater than or equal to data. SELECT CEIL(456.54) FROM DUAL; Output: 457
FLOOR(data)	It returns largest integer less than or equal to data. SELECT FLOOR(456.54) FROM DUAL; Output: 456
LN(data)	It returns natural logarithm of data.
LOG(b, data)	It returns base 'b' logarithm of data.
MOD(data, y)	It returns the modulus of dividing data by 'y'.
POWER(data, y)	It returns data raised to the power of 'y'.
ROUND(data, n)	It returns round value where 'n' is number of decimal place. SELECT ROUND(456.54, 0), ROUND(456.54, 1) FROM DUAL; Output: 457 456.5
SQRT(data)	It returns square root of data. SELECT SQRT(9) FROM DUAL; Output: 3
TRUNC(data, n)	It returns truncated value where 'n' is number of decimal places for truncation. SELECT TRUNC(456.54, 0), TRUNC(456.54, 1) FROM DUAL; Output: 456 456.5

Example: -

ABS

```
SQL> select abs (-345) from dual;
```

```
ABS(-345)
---------
      345
```

CEIL

```
SQL> select ceil (456.67) from dual;
```

```
CEIL(456.67)
------------
         457
```

FLOOR

```
SQL> select floor (456.34) from dual;
```

```
FLOOR(456.34)
-------------
          456
```

ROUND

```
SQL> select round (456.54,0), round (456.54,1), round (456.54,2) from
dual;
```

```
ROUND(456.54,0) ROUND(456.54,1) ROUND(456.54,2)
--------------- --------------- ---------------
            457           456.5          456.54
```

SQRT

```
SQL> select sqrt (9) from dual;
```

```
  SQRT(9)
---------
        3
```

TRUNC

```
SQL> select trunc (456.54 ,0), trunc (456.54 ,1) from dual;
```

```
TRUNC(456.54,0) TRUNC(456.54,1)
--------------- ---------------
            456           456.5
```

LN

```
SQL> select ln (5) from dual;
```

```
     LN(5)
----------
1.60943791
```

LOG

```
SQL> select log (2,5) from dual;
```

```
  LOG(2,5)
----------
2.32192809
```

MOD

```
SQL> select mod (2,5) from dual;
```

```
  MOD(2,5)
----------
         2
```

CONVERSION FUNCTIONS IN ORACLE

TO_DATE

This is used to convert any string format into date format.

```
SQL> insert into birth values ('Suman', to_date('2003-march-21',
'yyyy-mm-dd' ));
```

TO_CHAR

If we want to see only specific part.

```
SQL> select name,to_char (dob , 'mm') from birth;
```

```
NAME                                               TO
-------------------------------------------------- --
Suman                                              07
Suman                                              07
```

```
SQL> select name,to_char (dob , 'yyyy') from birth;
```

```
NAME                                               TO_C
-------------------------------------------------- ----
Suman                                              2009
Suman                                              2009
```

```
SQL> select name,dob from birth where to_char (dob,'q')='3';
```

```
NAME                                               DOB
-------------------------------------------------- ---------
Suman                                              23-JUL-09
Suman                                              23-JUL-09
```

(because July is in quarter 3)

Ouarter Division of Months
Q1 – January, February, March
Q2 – April, May, June
Q3 – July, August, September
Q4 – October, November, December

NVL

It fills default value in the null fields.

```
SQL> Select fieldname, fieldname nvl(fieldname , default_value) from
tablename;
```

Example: -

```
 ENO ENAME                          DNO     SALARY
---- -------------------- ---------- ----------
   1 Suman                          10      70000
   2 Tanu                           20      80000
   3 Ayan                           10      80000
   4 Joy                            50      75000
   5 Boja                           70
```

```
SQL> select eno,ename, eno, nvl(salary , 60000) from empl;
```

```
 ENO ENAME                        ENO NVL(SALARY,60000)
---- -------------------- ---------- -----------------
   1 Suman                         1             70000
   2 Tanu                          2             80000
   3 Ayan                          3             80000
   4 Joy                           4             75000
   5 Boja                          5             60000
```

DECODE

Decode Function is used to expand a small abbreviation.

```
SQL> select field, decode (field_to_be_decoded, 'small_name',
'full_name','NOT SPECIFIED')"new_column_name" from tablename;
```

Example: -

```
NAME                 CITY
-------------------- ---------
Suman                ADA
lalu                 DHN
Sumit                RNP
```

```
SQL> select name, city, decode (city, 'ADA', 'ADRA' , 'DHN' ,
'DHANBAD' , 'RNP' , 'RAGHUNATHPUR','NOT SPECIFIED')"DECODED CITY" from
city;
```

```
NAME                 CITY       DECODED CITY
-------------------- ---------- -------------
Suman                ADA        ADRA
lalu                 DHN        DHANBAD
Sumit                RNP        RAGHUNATHPUR
```

DATE FUNCTIONS

ADD_MONTHS(date, count)	It adds number months 'count' in the date. SELECT ADD_MONTHS('06-JUN-88',5) FROM DUAL; Output: 06-NOV-88
LAST_DAY(date)	It gives the last day of the specified month. SELECT LAST_DAY('06-JUN-88') FROM DUAL; Output: 30-JUN-88
MONTHS_BETWEEN(date1,date2)	It gives date1-date2 in months format. SELECT MONTHS_BETWEEN('06-JUN-88','06-NOV-88') FROM DUAL; Output: 5
NEXT_DAY(date, 'day')	It gives date of the next day after specified date. Where 'day' is Monday, Tuesday etc. SELECT NEXT_DAY('06-JUN-88','MON') FROM DUAL; Output: 13-JUN-88
TO_DATE(string)	It converts string into Oracle date format. INSERT INTO DATE VALUES ('kunal', TO_DATE('87-JAN-01','YY-MON-DD')); Output:
NEW_TIME(date,z1,z2)	It returns one zone time into other zone time. SELECT NEW_TIME('06-JUN-88','AST','CST') FROM DUAL; Output: 05-JUN-88

Example: -

ADD_MONTHS

```
SQL> select add_months('6-jun-88',5) from dual;
```

```
ADD_MONTH
---------
06-NOV-88
```

LAST_DAY

```
SQL> select last_day('6-jun-88') from dual;
```

```
LAST_DAY(
---------
30-JUN-88
```

NEXT_DAY

```
SQL> select next_day('6-jun-88','WED') from dual;
```

```
NEXT_DAY(
---------
08-JUN-88
```

MONTHS_BETWEEN

```
SQL> select months_between ('6-jun-88','6-nov-88') from dual;
```

```
MONTHS_BETWEEN('6-JUN-88','6-NOV-88')
-------------------------------------
                                   -5
```

NEW_TIME

```
SQL> select new_time('6-jun-88','gmt','bst') from dual;
```

```
NEW_TIME(
---------
05-JUN-88
```

SQL COMMAND TYPES IN ORACLE

- **Data Definition Language (DDL)**

  DDL is used to create and remove database objects.

  Example: CREATE, ALTER, DROP commands
- **Data Manipulation Language (DML)**

  DML is used to manipulate data in the database.

  Example: INSERT, DELETE, UPDATE, SELECT commands
- **Data Control Language (DCL)**

  DCL is used to control the kind of data transaction and access to the database, like who can access and who can not access.

  Example: COMMIT, ROLLBACK are transaction control commands. GRANT, REVOKE are data access commands.

SQL BUFFER COMMANDS

Buffer Stores the Number of Lines in a query.

```
SQL> select *
  2  from
  3  empl
  4  ;
```

TO SEE THE LINES

SQL> L;

```
SQL> L
  1  select *
  2  from
  3  empl
  4*
```

TO SELECT A LINE AS THE NEWLINE

SQL> *line_number;*

Example: -

SQL> 1

```
SQL> 1
  1* select *
```

(1 is selected as the new line)

INSERT A LINE AFTER THE NEW LINE

SQL> Input *input_statement;*

Example: -

(1 is selected as the new line)

SQL> Input this is the inserted new line

```
SQL> 1
  1* select *
SQL> input this is the inserted new line
SQL> l
  1  select *
  2  this is the inserted new line
  3  from
  4  empl
  5*
```

DELETE A LINE

Select the Line and

SQL> del

Example: -

SQL> 2 **(Selects 2 as the new line)**

SQL> del **(Deletes the line)**

```
SQL> 2
  2* this is the inserted new line
SQL> del
SQL> l
  1  select *
  2  from
  3  empl
  4*
```

APPEND IN A LINE

Select the line and

SQL> Append, *append_string*

Example: -

SQL> 1 **(Selects 1 as the new line)**

SQL> Append, from empl

```
SQL> 1
  1* select *
SQL> append, from empl
  1* select *, from empl
SQL> l
  1  select *, from empl
  2  from
  3  empl
  4*
```

CHANGE SPECIFIC STRING IN THE QUERY

Select the line and

SQL> Change / *string_to_be_changed* / *changed_string*

Example: -

SQL>1 **(Selects the line 1 as the newline)**

SQL> Change / *, from empl / id , name

```
SQL> 1
  1* select*, from empl
SQL> change /*, from empl/id,name
  1* selectid,name
SQL> l
  1  selectid,name
  2  from
  3  empl
  4*
```

RUN THE BUFFER COMMAND

SQL> /

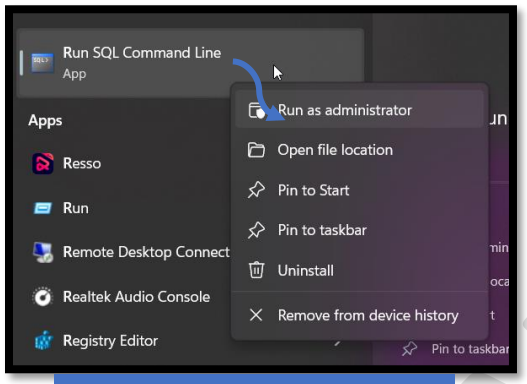
ED OR EDIT BUFFER

First, we have to open the 'run sql command line' as an administrator.

Now, `SQL> ed`

(It will create a buffer notepad file …sekhane amar run kora query ta lekha thakbe…ami notepad e kichu change korle seta command line e execute hobe…command execute korar jonno '/' use korte hobe)

Example: -



STEP 1 – Run as Administrator

```
SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL> _
```

STEP 2 – Connect

```
SQL> select *from empl;

       ENO ENAME                      DNO     SALARY
---------- -------------------- ---------- ----------
         1 Suman                        10      70000
         2 Tanu                         20      80000
         3 Ayan                         10      80000
         4 Joy                          50      75000
         5 Boja                         70

SQL> ed
Wrote file afiedt.buf
```

afiedt.buf - Notepad

File Edit View

```
select *from empl
/
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

STEP 3 – Execute any Command and then, SQL> ed. Now a buffer notepad file will be created

afiedt.buf - Notepad

File Edit View

```
select *from dept
/
```

Ln 1, Col 18 | 100% | Windows (CRLF) | UTF-8

STEP 4 – Change in the notepad file

```
  1* select *from dept
SQL> /

       DNO DNAME
---------- --------------------
        10 MCA
        20 MBA
        30 BCA
        40 MTECH
```

STEP 5 – Same Change will be shown in the command line and execute it using ' / '.

NOT NULL CONSTRAINT

If we create a field as not null then we can't insert null value in that. We can make multiple columns as not null.

```
SQL> Create table tablename (fieldname datatype not null);
```

Example: -

```
SQL> create table demo1 (id number, name varchar2(15) not null);
```

[Here the name field is not null]

```
SQL> insert into demo1 values(null,'Shyam');

1 row created.
```

We can store null value in ID

```
SQL> insert into demo1 values(2,null);
insert into demo1 values(2,null)
                          *
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."DEMO1"."NAME")
```

But we can't store null value in name

UNIQUE CONSTRAINTS

If we make a column as unique then we have to store unique values for each field in that column. We can make multiple columns unique.

```
SQL> Create table tablename (fieldname datatype unique);
```

Example: -

```
SQL> create table demo (id number unique, name varchar2(15));
```

[Here the ID field is unique]

```
SQL> insert into demo values (1, 'Suman');

1 row created.

SQL> insert into demo values (2, 'Suman');

1 row created.
```

The Name field can store duplicate values

```
SQL> insert into demo values (2, 'Suman');

1 row created.

SQL> insert into demo values (2, 'Tanu');
insert into demo values (2, 'Tanu')
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004089) violated
```

The ID field can't store duplicate values

PRIMARY KEY CONSTRAINT

Primary key identifies each record of the column uniquely.

A Primary Key –

- Accepts Only Unique Values.
- Cannot be Null.
- A table can have only one primary key.

```
SQL> Create Table tablename (fieldname datatype primary key);
```

Example: -

```
SQL> create table parent (parent_id number primary key, parent_name
varchar2(15) not null, city varchar2(10));
```

```
Name                                      Null?    Type
----------------------------------------- -------- ----------------
PARENT_ID                                 NOT NULL NUMBER
PARENT_NAME                               NOT NULL VARCHAR2(15)
CITY                                               VARCHAR2(10)
```

Primary Key is Not Null

```
SQL> insert into parent values(2, 'Shyam' , 'Adra');

1 row created.
```

Unique value can be inserted

```
SQL> insert into parent values(2, 'Raja' , 'bankura');
insert into parent values(2, 'Raja' , 'bankura')
*
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C004091) violated
```

But it can't store duplicate values

FOREIGN KEY

Foreign key is used to make relationship between two tables. The foreign key of a table points to the primary key of another table.

A Foreign key –

- Can be Null.
- Can have duplicate values.
- It accepts only those value which are available on the parent / base table

We can make foreign key in two methods –

(Method-1)

```
SQL> Create Table tablename (fieldname (same as parent table primary key) references parent_table_name);
```

Example: -

```
SQL> create table child (child_id number primary key, parent_id number references parent, toy varchar2(15));
```

[The 'parent' table is discussed in the primary key discussion section]

```
Name                                      Null?    Type
----------------------------------------- -------- ------------------
CHILD_ID                                  NOT NULL NUMBER
PARENT_ID                                          NUMBER
TOY                                                VARCHAR2(15)
```

```
PARENT_ID PARENT_NAME     CITY
---------- --------------- ----------
         1 Ram             Adra
         2 Shyam           Adra
```

This is the Parent Table

```
SQL> insert into child values (1 , 1 , 'Video Game');

1 row created.
```

We can insert the values of ‘parent_id’ which are available in the parent table (Ex:- 1)

```
SQL> insert into child values (2 , 3 , 'Bow');
insert into child values (2 , 3 , 'Bow')
*
ERROR at line 1:
ORA-02291: integrity constraint (SYSTEM.SYS_C004093) violated - parent key not
found
```

We can’t insert the values of ‘parent_id’ which are not available in the parent table (Ex:- 3)

(Method 2)

Normally Make a table without referencing

Now,

```
SQL> alter table child_table_name add constraint constraint_name
foreign key (child_table_fieldname) references parent_table_name
(parent_table_fieldname);
```

We can Drop the Constraint to remove the Foreign Key using

```
SQL> alter table child_table_name drop constraint constraint_name;
```

Example: -

```
SQL> create table child(child_id number primary key, parent_id number,
toy varchar2(15));
```

[Normally Creating a Table]

```
PARENT_ID PARENT_NAME     CITY
---------- --------------- ----------
         1 Ram             Adra
         2 Shyam           Adra
```

This is the Parent Table

```
SQL> alter table child add constraint fk_parentchild foreign
key(parent_id) references parent(parent_id);
```

```
SQL> insert into child values (1 , 1 , 'Video Game');

1 row created.
```

We can insert the values of 'parent_id' which are available in the parent table (Ex:- 1)

```
SQL> insert into child values (2 , 3 , 'Bow');
insert into child values (2 , 3 , 'Bow')
*
ERROR at line 1:
ORA-02291: integrity constraint (SYSTEM.SYS_C004093) violated - parent key not
found
```

We can't insert the values of 'parent_id' which are not available in the parent table (Ex:- 3)

Dropping the Constraint

```
SQL> alter table child drop constraint fk_parentchild;
```

```
SQL> insert into child values(2,3,'hg');

1 row created.
```

Now we can insert the values of 'parent_id' which are not available in the parent table (Ex:- 3)

CHECK CONSTRAINT

We use this constraint in create table and alter table. We use this to check certain conditions on a column.

(Check Constraint in Create Table)

```
SQL> Create Table tablename (fieldname datatype, constraint
constraint_name check (condition));
```

Example: -

(1)

```
SQL> create table marks (name varchar2(15), marks number, constraint
check_marks check (marks between 0 and 100));
```

```
SQL> insert into marks values('Suman' , 89);

1 row created.
```

We can insert the value of marks which are between 0 and 100

```
SQL> insert into marks values('Suman' , 120);
insert into marks values('Suman' , 120)
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.CHECK_MARKS) violated
```

We can't insert the value of marks which are not between 0 and 100

(2)

```
SQL> create table marks (name varchar2(15), marks number, constraint
check_name check (name = upper(name)));
```

```
SQL> insert into marks values('SUMAN' , 92);

1 row created.
```

We can insert name in Uppercase

```
SQL> insert into marks values('Suman' , 96);
insert into marks values('Suman' , 96)
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.CHECK_NAME) violated
```

We cannot insert name without Uppercase

(Check Constraint in Alter Table)

Normally Create a Table

Now,

```
SQL> alter table tablename add constraint constraint_name check
(condition)
```

Example: -

```
SQL> create table marks(name varchar2(15), marks number);
```

[Normally Creating a Table]

```
SQL> alter table marks add constraint check_marks check (marks between
0 and 100);
```

```
SQL> insert into marks values('Suman' , 89);

1 row created.
```

We can insert the value of marks which are between 0 and 100

```
SQL> insert into marks values('Suman' , 120);
insert into marks values('Suman' , 120)
*
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.CHECK_MARKS) violated
```

We can't insert the value of marks which are not between 0 and 100

Temporarily Disable the Constraint: -

```
SQL> alter table tablename disable constraint constraint_name;
```

Example: -

```
SQL> alter table marks disable constraint check_marks;
```

```
SQL> insert into marks values('Suman' , 109);

1 row created.
```

Now we can insert the value of marks which are not between 0 and 100

Re Enable Constraint: -

```
SQL> alter table tablename disable constraint constraint_name;
```

[For re enabling we have to delete such records which are violating the condition of the constraint]

Drop the Constraint Permanently: -

```
SQL> alter table tablename drop constraint constraint_name;
```

GROUP BY

Group by clause is used in select statement. It is used for the grouping of multiple records. It usually has an aggregate function but this is not mandatory.

```
SQL> Select fieldname from tablename group by fieldname.
```

Example: -

```
PRODUCT               SALE
--------------- ----------
mouse                  200
keyboard               300
cabinet                900
mouse                  400
cabinet               1200
keyboard               600
mouse                  200
cabinet               1100
keyboard               450
keyboard               450
```

```
SQL> select product from orders group by product;
```

[It groups the products]

```
PRODUCT
---------------
keyboard
cabinet
mouse
```

```
SQL> select product,count(sale) from orders group by product;
```

[It counts the number of sale of a product]

```
PRODUCT         COUNT(SALE)
--------------- -----------
keyboard                  4
cabinet                   3
mouse                     3
```

```
SQL> select product,sum(sale) from orders group by product;
```

[It counts the total sale of a product]

```
PRODUCT          SUM(SALE)
--------------- ----------
keyboard              1800
cabinet               3200
mouse                  800
```

```
SQL> select product,min(sale) from orders group by product;
```

[It displays the minimum price of a product]

```
PRODUCT          MIN(SALE)
--------------- ----------
keyboard               300
cabinet                900
mouse                  200
```

```
SQL> select product,max(sale) from orders group by product;
```

[It displays the maximum price of a product]

```
PRODUCT          MAX(SALE)
--------------- ----------
keyboard               600
cabinet               1200
mouse                  400
```

HAVING CLAUSE

Having clause is used in select statement with group by clause. It is used to apply conditions in the group by statement.

```
SQL> Select fieldname from tablename group by fieldname having
condition.
```

Example: -

```
PRODUCT                 SALE
--------------- ----------
mouse                    200
keyboard                 300
cabinet                  900
mouse                    400
cabinet                 1200
keyboard                 600
mouse                    200
cabinet                 1100
keyboard                 450
keyboard                 450
```

(1) `SQL> select product, sum(sale) from orders group by product having sum(sale)>2000;`

```
PRODUCT          SUM(SALE)
--------------- ----------
cabinet               3200
```

[The product which has a sale of greater than rupees 2000 will be selected]

(2) `SQL> select product, count(sale) from orders group by product having count(sale)>=4;`

```
PRODUCT         COUNT(SALE)
--------------- -----------
keyboard                  4
```

[The product which has a sale of greater or equal than 4 times will be selected

SUBQUERY OR INNER QUERY OR NESTED QUERY

Inside a query there is another query then it is called subquery or nested query or inner query.

In oracle we can use 255 level subqueries with where clause.

Example: -

(1)

```
 ENO ENAME                     DNO     SALARY
----- -------------------- ---------- ----------
    1 Suman                        10      70000
    2 Tanu                         20      80000
    3 Ayan                         10      80000
    4 Joy                          50      75000
    5 Boja                         70
```

```
SQL> select *from empl where eno = (select max(eno) from empl);
```

Explanation: - The subquery 'select max(eno) from empl' returns the maximum eno of the table, which is 5.

Now the query returns all the details of the employee whose eno is 5.

```
 ENO ENAME                     DNO     SALARY
----- -------------------- ---------- ----------
    5 Boja                         70
```

(2)

```
PRODUCT                 SALE
--------------- ----------
mouse                    200
keyboard                 300
cabinet                  900
mouse                    400
cabinet                 1200
keyboard                 600
mouse                    200
cabinet                 1100
keyboard                 450
keyboard                 450
```

```
SQL> select *from orders where product in (select product from orders
where sale>400);
```

Explanation: – The Subquery returns the products whose price is above 400.

They are,

```
PRODUCT
----------
cabinet
cabinet
keyboard
cabinet
keyboard
keyboard
```

Now the query returns all the products of this names with their details.

```
PRODUCT                 SALE
--------------- ----------
cabinet                 1100
cabinet                 1200
cabinet                  900
keyboard                 450
keyboard                 450
keyboard                 600
keyboard                 300
```

UNION OPERATOR

It merges two tables and removes the duplicates (if present).

```
SQL> select *from table1 union select *from table2;
```

Example: -

```
  ID NAME                 SALARY
---- --------------- ----------
   1 Suman                30000
   2 Ayan                 50000
   3 Tanumoy              50000
```

emp2013

```
  ID NAME                 SALARY
---- --------------- ----------
   4 Rohan                60000
   5 Soham                45000
   3 Tanumoy              50000
```

emp2014

```
SQL> select *from emp2013 union select *from emp2014;
```

```
  ID NAME                 SALARY
---- --------------- ----------
   1 Suman                30000
   2 Ayan                 50000
   3 Tanumoy              50000
   4 Rohan                60000
   5 Soham                45000
```

It removed the Duplicate

UNION ALL

It merges two tables and doesn't remove the duplicates (if present).

```
SQL> select *from table1 union all select *from table2;
```

Example: -

```
  ID NAME                SALARY
---- --------------- ----------
   1 Suman                30000
   2 Ayan                 50000
   3 Tanumoy              50000
```

emp2013

```
  ID NAME                SALARY
---- --------------- ----------
   4 Rohan                60000
   5 Soham                45000
   3 Tanumoy              50000
```

emp2014

```
SQL> select *from emp2013 union all select *from emp2014;
```

```
  ID NAME                SALARY
---- --------------- ----------
   1 Suman                30000
   2 Ayan                 50000
   3 Tanumoy              50000
   4 Rohan                60000
   5 Soham                45000
   3 Tanumoy              50000
```

It does not remove the duplicates.

INSTERSECT

It returns the tuples which are present in both of the tables.

```
SQL> select *from table1 intersect select *from table2;
```

Example: -

```
  ID NAME                 SALARY
---- --------------- ----------
   1 Suman                30000
   2 Ayan                 50000
   3 Tanumoy              50000
```

emp2013

```
  ID NAME                 SALARY
---- --------------- ----------
   4 Rohan                60000
   5 Soham                45000
   3 Tanumoy              50000
```

emp2014

```
SQL> select *from emp2013 intersect select *from emp2014;
```

```
  ID NAME                 SALARY
---- --------------- ----------
   3 Tanumoy              50000
```

MINUS

It selects the tuples which are present in the first table but not in the second table.

```
SQL> select *from table1 minus select *from table2;
```

Example: -

```
 ID NAME                SALARY
---- --------------- ----------
   1 Suman                30000
   2 Ayan                 50000
   3 Tanumoy              50000
```

emp2013

```
 ID NAME                SALARY
---- --------------- ----------
   4 Rohan                60000
   5 Soham                45000
   3 Tanumoy              50000
```

emp2014

```
SQL> select *from emp2013 minus select *from emp2014;
```

```
 ID NAME                SALARY
---- --------------- ----------
   1 Suman                30000
   2 Ayan                 50000
```

3 Tanumoy 50000

Is present in the seond table. So it is removed.

EXECUTE COMMANDS WRITTEN IN PHYSICAL FILE

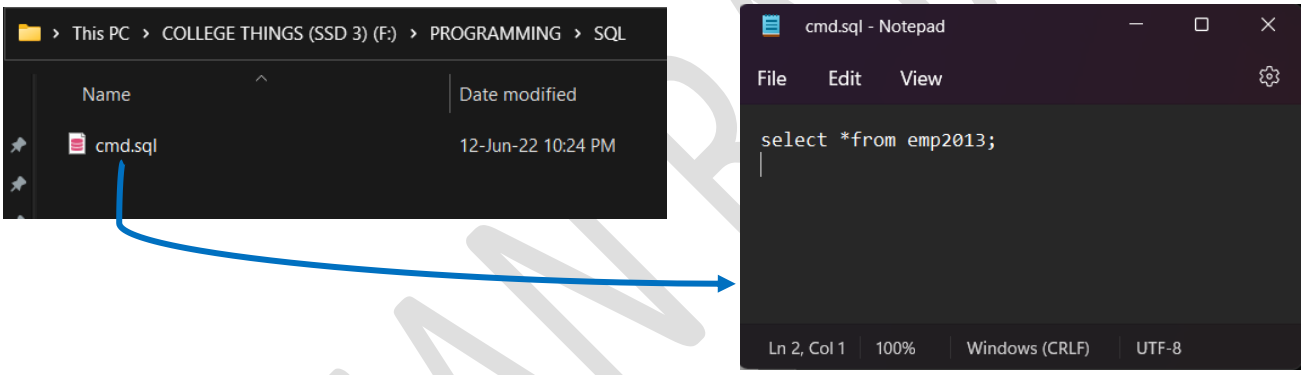
It runs the sql query which is present in any physical file in '.sql' or '.txt' format.

Make a .sql or .txt file in notepad.

Now,

```
SQL> Start file location (use '/') / filename;
```

Example: -




```
SQL> start F:/PROGRAMMING/SQL/cmd.sql;
```

```
ID NAME                 SALARY
--- --------------- ----------
  1 Suman                30000
  2 Ayan                 50000
  3 Tanumoy              50000
```

MAKE NEW FILE USING COMMAND LINE

```
SQL> Edit newfilename.extension(txt or sql);
```

Now save the file and run using the previous method using ‘Start’ command.