

SQL

ORACLE

COMPLETE

@Suman Rajak





15-16

14-15

Pseudo Columns

13

BETWEEN Operator

12

IN Operator

10-11

Like Pattern Operator

9

To Display without a Specific Records

To Order using Multiple Fields

9

8-9

Order By

Select with Multiple Condition

Select Null Statement

8

7

Concatenate Using Select Command

Select Command with Where Condition

7

7

Update Records in the Table

6

Delete Records of the Table

5-6

Datatypes in ORACLE

2

Clear the Screen

1

Connect to the Database

1

1

What is SQL

TABLE OF CONTENTS

2-3

Create Table

3

Show Description of the Table

3-4

Insert Values in the Table

5

Show Records of the Table

Currval and Nextval



18-19

Roll Back

Drop Table

18

Rename Table

18

RowID and RowNum

17

32

Table Alias

Self Join

34

Natural Join

20

Commit

21

Creating Tablespace

21

Creating User

21

Creating Role



32-33

30-31

Non Equi Join

Referential Integrity or Foreign Key

Equi Join

29

Drop User

27

Drop Roles

27

Column Alias

27

28

Grant a user table to another table

26

24

Drop Tablespace

25

Insert Multiple Records in One or More Table at Once

24

Remove Datafile from a Tablespace

24

See Total Tablespace List

See Total Datafiles List

23

23

23

Add New Datafile to an Existing Tablespace

Alter Tablespace

Assigning Permission to Role

22

Passing Role to User

22



Conversion Functions in Oracle

48

NVL

49

Decode

SQL Buffer Commands

56

Not Null Constraint

68-69

Subquery or Inner Query or Nested Query

70

71

Union All Operator

Union Operator

72

Intersect

35

Cross Join

36

Left Outer Join

37

Right Outer Join

38

Full Outer Join

39

Anti Join or Not IN

40

Column Format

41

Clear Column Formatting

Aggregate Functions

Character Functions

Number Functions

Date Functions

51

SQL Command Types in Oracle

57

Unique Constraint

Check Constraint

Group By

58

Primary Key Constraint

Foreign Key

67

Having Clause



65-66

62-65

59-62

52-55

50-51

47-48

45-46

43-44

41-43



Make New File Using Command Line

75

Execute Commands Written in Physical File

74

Minus

73

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:

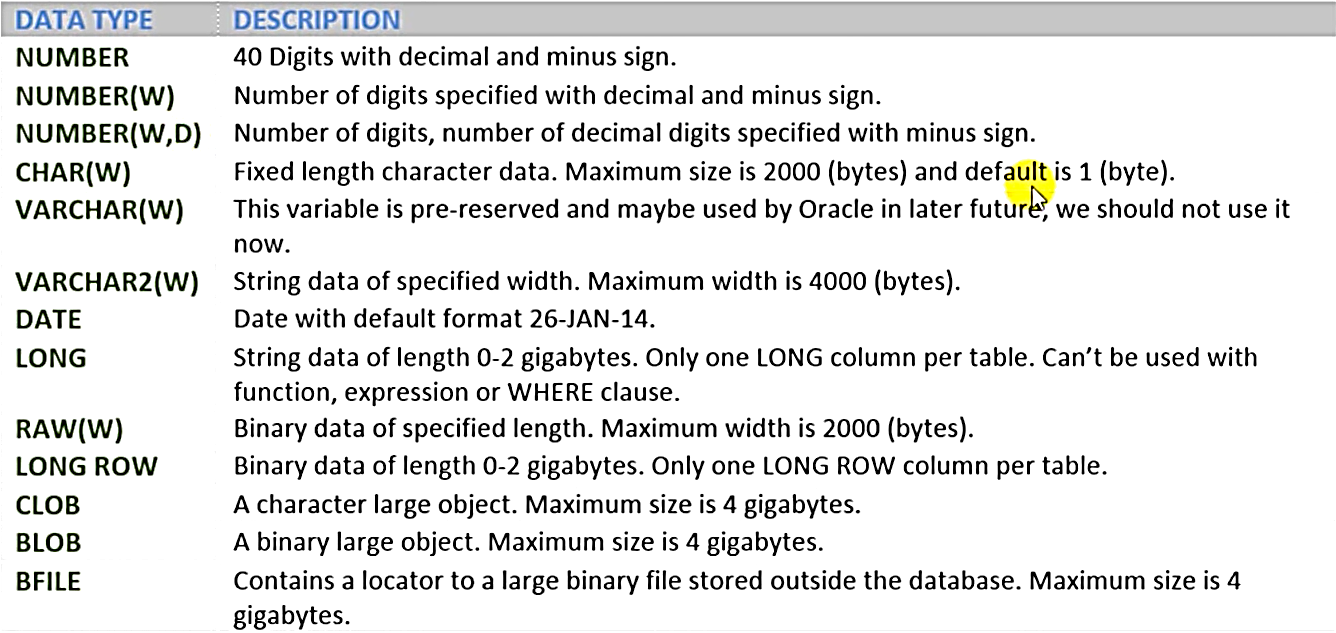
**Alternative method: -**

SQL> connect *username*/*password*;

CLEAR THE SCREEN

SQL> clear screen;

DATATYPES IN ORACLE SQL



CREATE TABLE

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

Condition:-

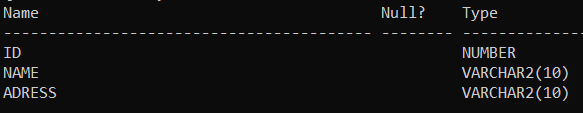
* It is a DDL Command
* Table name can be upto 30 char long.
* Table name must begin with alphabet.
* Table name can’t contain single or double quote.
* Names are not case sensitive.
* Names can contain a-z , 0-9, \_ , $ , #
* Names can’t be reserved words.

Command: -

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

**Example: -**

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



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: -

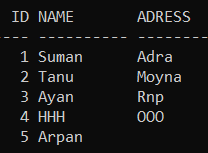
Enter the value of Address: -

Insert Specific fields into the table

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

**Example: -**

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

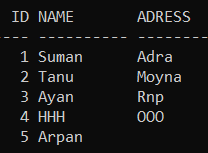


SHOW THE RECORDS OF THE TABLE

**Show the full table: -**

SQL> Select \*from *tablename;*

**Example: -**

****Select \*from Student;

**Show specific fields of the table: -**

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

**Example: -**

Select Id, Name from Student;

DELETE RECORD FROM TABLE

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

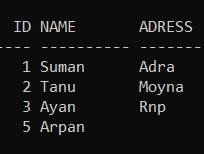
**Delete all records: -**

SQL> Delete from *tablename*;

**Delete specific records with a condition: -**

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

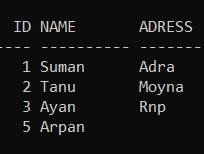
**Example: -**

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

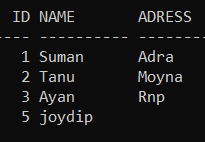
UPDATE RECORDS IN THE TABLE

Updates records in the table

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

**Example: -**

SQL> Update Student2 set name= ‘joydip’ where name = ‘Arpan’;

****

SELECT COMMAND WITH WHERE CONDITION

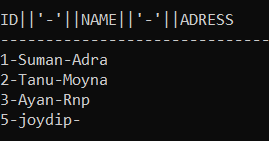
**For all the fields: -**

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

Select Null Statement

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

**Example: -**

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

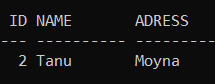
****

Select with Multiple Conditions

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

Example: -

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



ORDER BY

It Orders the Output of in ascending or descending order.

**For Ascending Order: -**

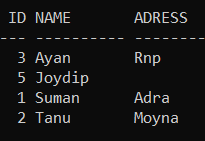
SQL> Select \*from *tablename* order by *fieldname;*

**For Descending Order: -**

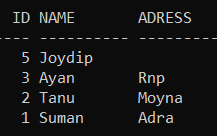
SQL> Select \*from *tablename* order by *fieldname* desc;

**Example: -**

1. SQL> Select \*from student2 order by name;



1. SQL> Select \*from student2 order by id desc;



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;*

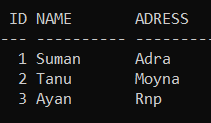
If there is some matching field in field1 then we will order using the field2

To display without a specific record

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

**Example: -**

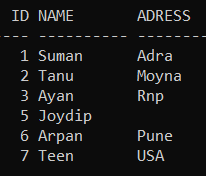
SQL> Select \*from student2 where not id=5;



LIKE ‘PATTERN’ OPERATOR

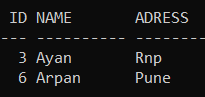
**Pattern: -**

* % :- 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: -**

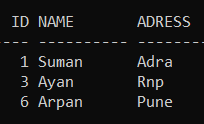
**First Letter: -**

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

****

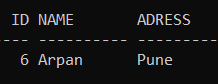
**Last Letter: -**

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



**Second Letter: -**

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



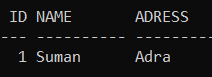
**Middle Letter: -**

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



**Third Letter: -**

SQL> Select \*from student2 where name like '\_\_m%’;



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’*

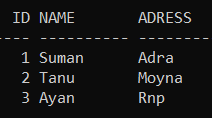
**Example: -**

**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’;

****

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;*

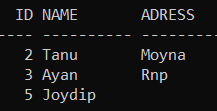
**Example: -**

**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;



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

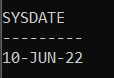
USER – UserID of current user

CURRVAL – current value

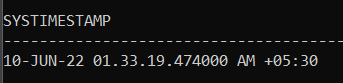
NEXTVAL – next value

**Example: -**

**SYSDATE: -**

SQL> select sysdate from dual;

**SYSTIMESTAMP: -**

SQL> select systimestamp from dual;

**UID: -**

SQL> select uid from dual;

**UID: -**

SQL> select user from dual;

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

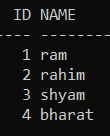
5 INCREMENT BY 1

1. 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));

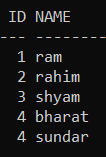


SQL> insert into tb1 vlaues (id\_seq.nextval , 'ram');

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');



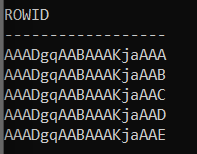
SQL> insert into tb1 values (id\_seq.currval , 'sundar');

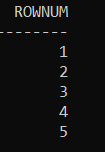
ROWID AND ROWNUM

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.

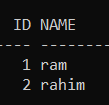
**Example: -**

****SQL> select rowid from tb1;

SQL> select rownum from tb1;

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

**Ans:-**

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

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;

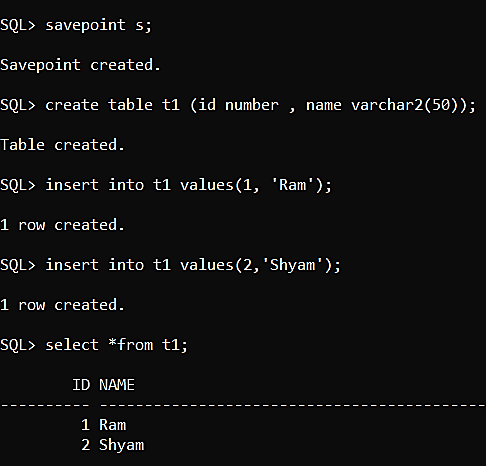
**STEP 1 🡪** SQL> Savepoint *savepoint\_name;*

**STEP 2 🡪** NOW MAKE SOME CHANGES IN THE DATABASE.

**STEP 3 🡪** SQL> Rollback to *savepoint\_name;*

**STEP 4 🡪** THE CHANGES WILL BE REMOVED.

**Example: -**

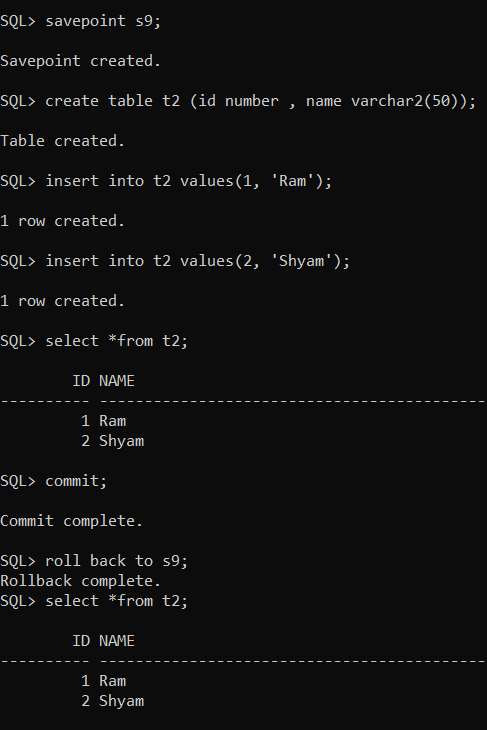
****

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: -**



CREATING TABLESPACE

SQL>CONNECT *username*/*password*;

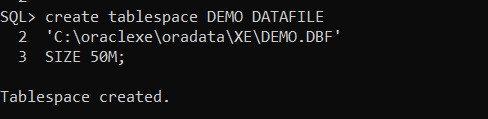
SQL>CREATE TABLESPACE *TablespaceName* DATAFILE

2 ‘C://oraclexe/oradata/xe/*TablespaceName*.dbf’

Will create a tablespace

Of size 50 MB

3 SIZE 50M;



CREATING USER

SQL>CREATE USER *UserName* IDENTIFIED BY *Password*

DEFAULT TABLESPACE *TablespaceName*

TEMPORARY TABLESPACE Temp

QUOTA UNLIMITED ON *TablespaceName;*

CREATING ROLE

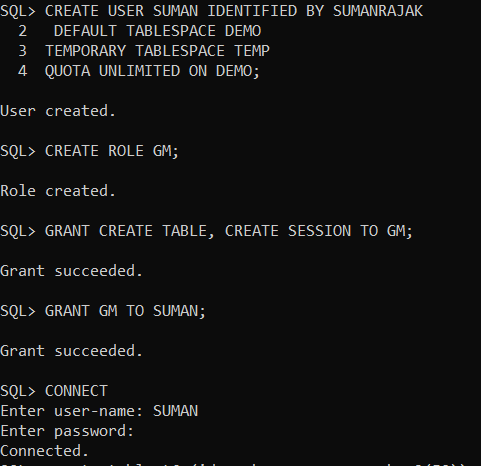
SQL>CREATE ROLE *RoleName*

ASSIGNING PERMISSION ON ROLE

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

PASSING ROLE TO USER

SQL>GRANT *RoleName* TO *UserName*;

**Example: -**

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.

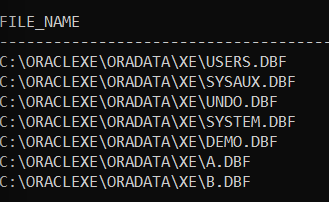
Add New Datafile to an Existing Tablespace

SQL> alter tablespace *tablespacename* add datafile 'C:\oraclexe\oradata\XE\*datafilename*.DBF'

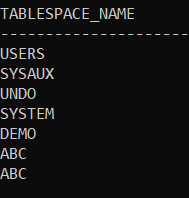
2 SIZE *size in mb*;

See Total Datafiles List

SQL> select file\_name from dba\_data\_files;



See Total tablespace list

SQL> select tablespace\_name from dba\_data\_files;

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 ()

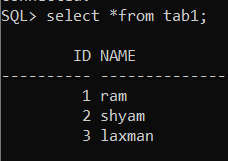
3 into *tablename* values ()

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

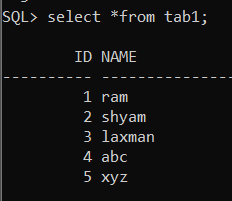
2 into tab1 values (1, ‘ram’)

3 into tab1values (2,’shyam’)

4 into tab1 values (3,’laxman’)

5 select \*from dual;

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

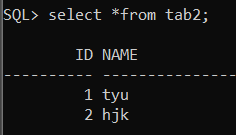
 SQL> insert all

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;

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));

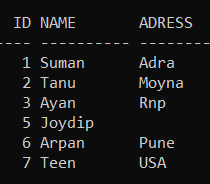
Because of ‘on delete cascade’ if we delete a record from the parent table then it will be automatically deleted from the child table.

COLUMN ALIAS

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

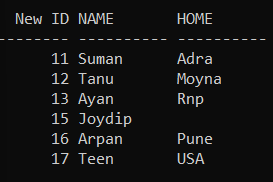
SQL> select *fieldname* "*desired field name in output*" from *tablename*;

**Example: -**



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

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

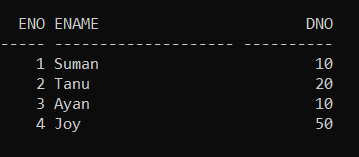


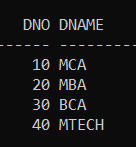
In this way, we can do calculations inside the query.

EQUI JOIN

In this join we use the equal operator.

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

Example: -

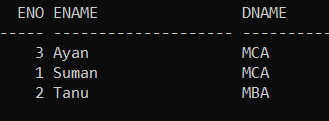


EMPL

DEPT

Between these two tables we will fetch the matching fields.

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

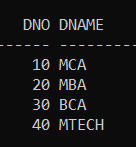
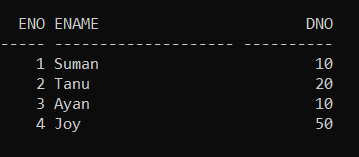


NON EQUI JOIN

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

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

**Example: -**

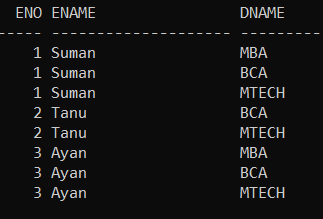
1. 

EMPL

DEPT

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;



In dept table,

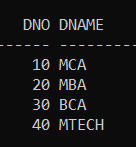
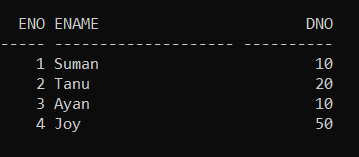
10 is greater than no one in the empl table.

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.

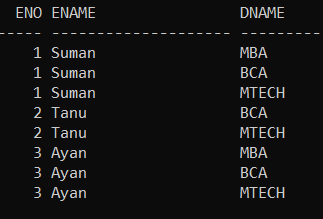
(2)

EMPL

DEPT

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;



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;

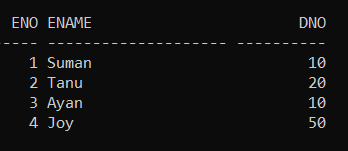
TABLE ALIAS i.e.; Table Temporary New Names

SELF JOIN

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

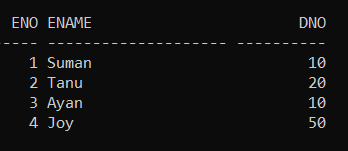
SQL> Select *tablenamealias.fieldname , tablenamealias.fieldname* from *tablename tablenamealias1 , tablename tablenamealias2* where *tablenamealias1.fieldname condition tablenamealias2.fieldname.*

**Example: -**

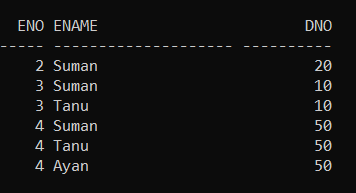
****

EMPL

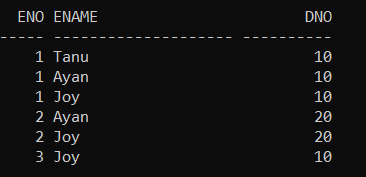
SQL> select e1.eno, e2.ename, e1.dno from empl e1 , empl e2 where e1.eno = e2.eno;



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



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

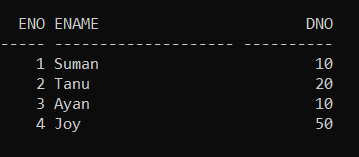
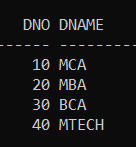


NATURAL JOIN

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

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

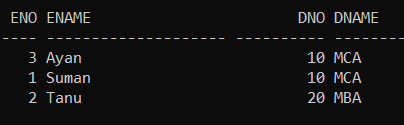
**Example: -**

****

DEPT

EMPL

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



CROSS JOIN

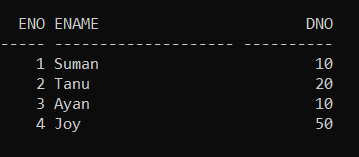
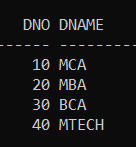
In the output the rows are merged of both of the tables without any condition

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

OR,

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

**Example: -**



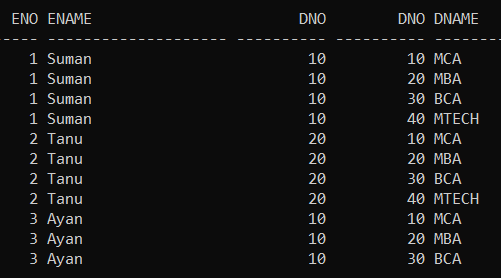
DEPT

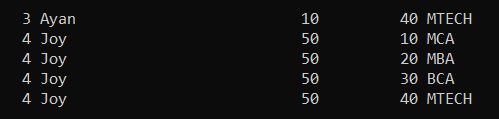
EMPL

SQL> select \*from empl, dept;

OR,

SQL> select \*from empl cross join dept;





LEFT OUTER JOIN

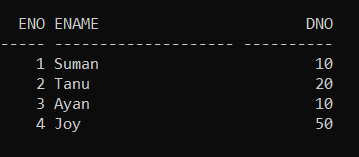
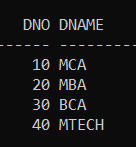
It returns all the tuples of the left relation and only the matching tuples from the right relation.

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: -**



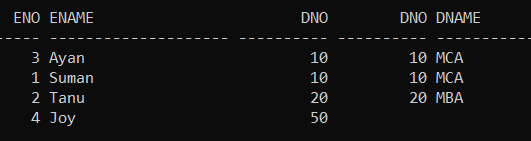
EMPL

DEPT

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

OR,

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



RIGHT OUTER JOIN

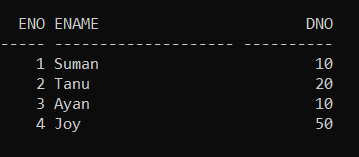
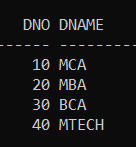
It returns all the tuples of the right relation and only the matching tuples from the left relation.

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: -**



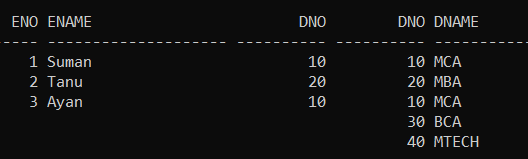
DEPT

EMPL

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

**OR,**

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

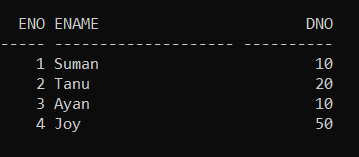
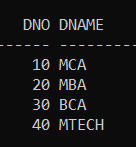


FULL OUTER JOIN

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*

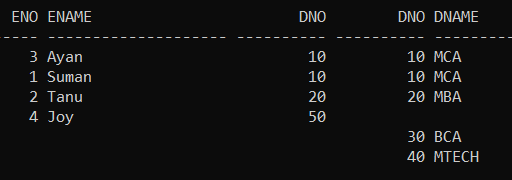
**Example: -**



DEPT

EMPL

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

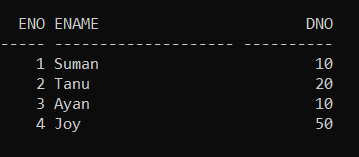
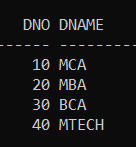


ANTI JOIN OR NOT IN OPERATOR

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);*

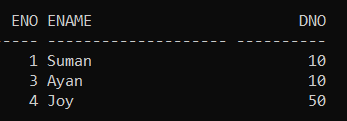
**Example: -**



EMPL

DEPT

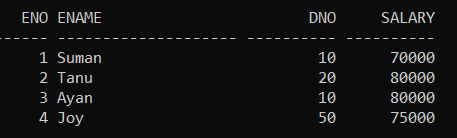
SQL> select \*from empl where dno not in (select dno from dept where dname='MBA');



COLUMN FORMAT

We can temporarily format any column as per our requirement.

SQL> Column *column\_name* format *format\_statement*;

**Example: -**

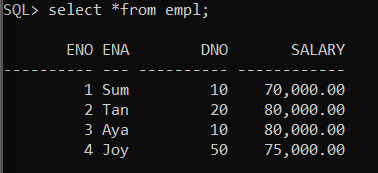
SQL> column salary format 9,99,999.99;

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



SQL> column ename format a3 trunc;

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



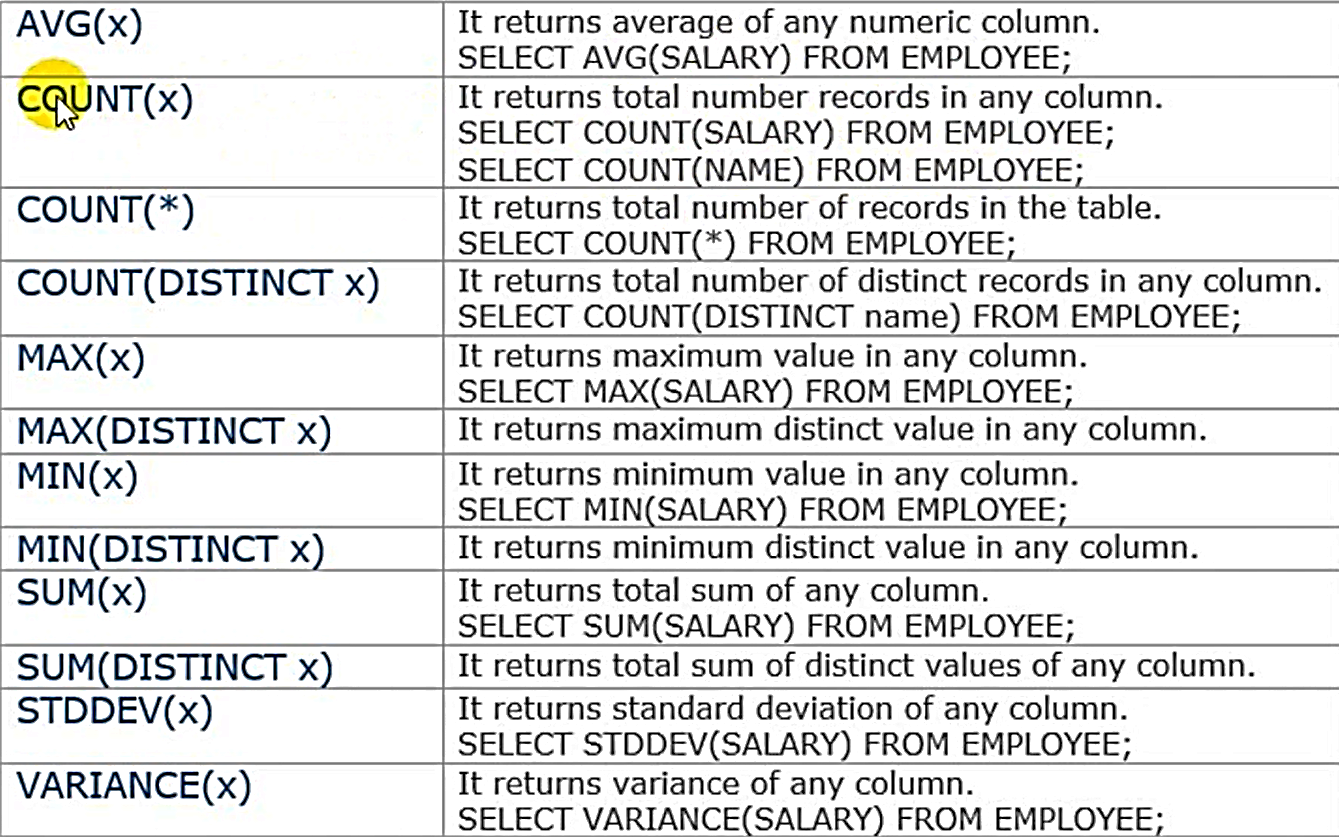
CLEAR COLUMN FORMATTING

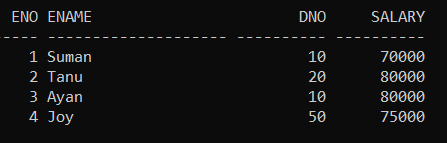
SQL> clear column;

OR,

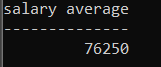
Restart the ‘Run SQL Command Line’

AGREEGATE FUNCTIONS IN ORACLE SQL



**Example: -**

1. **AVG**



SQL> Select avg (salary) from empl;

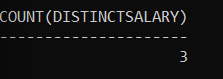
1. **COUNT (X)**

SQL> select count(salary) from empl;

1. **COUNT (\*)**

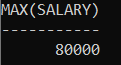
**** SQL> select count(\*) from empl;

1. **COUNT (DISTINCT X)**

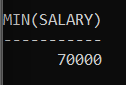


SQL> select count (distinct salary) from empl;

1. **MAX(X)**

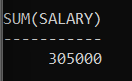
SQL> select max(salary) from empl;

1. **MIN (X)**

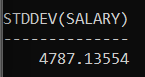


SQL> select min(salary) from empl;

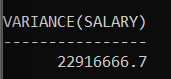
1. **SUM (X)**

SQL> select sum(salary) from empl;

1. **STDDEV (X)**

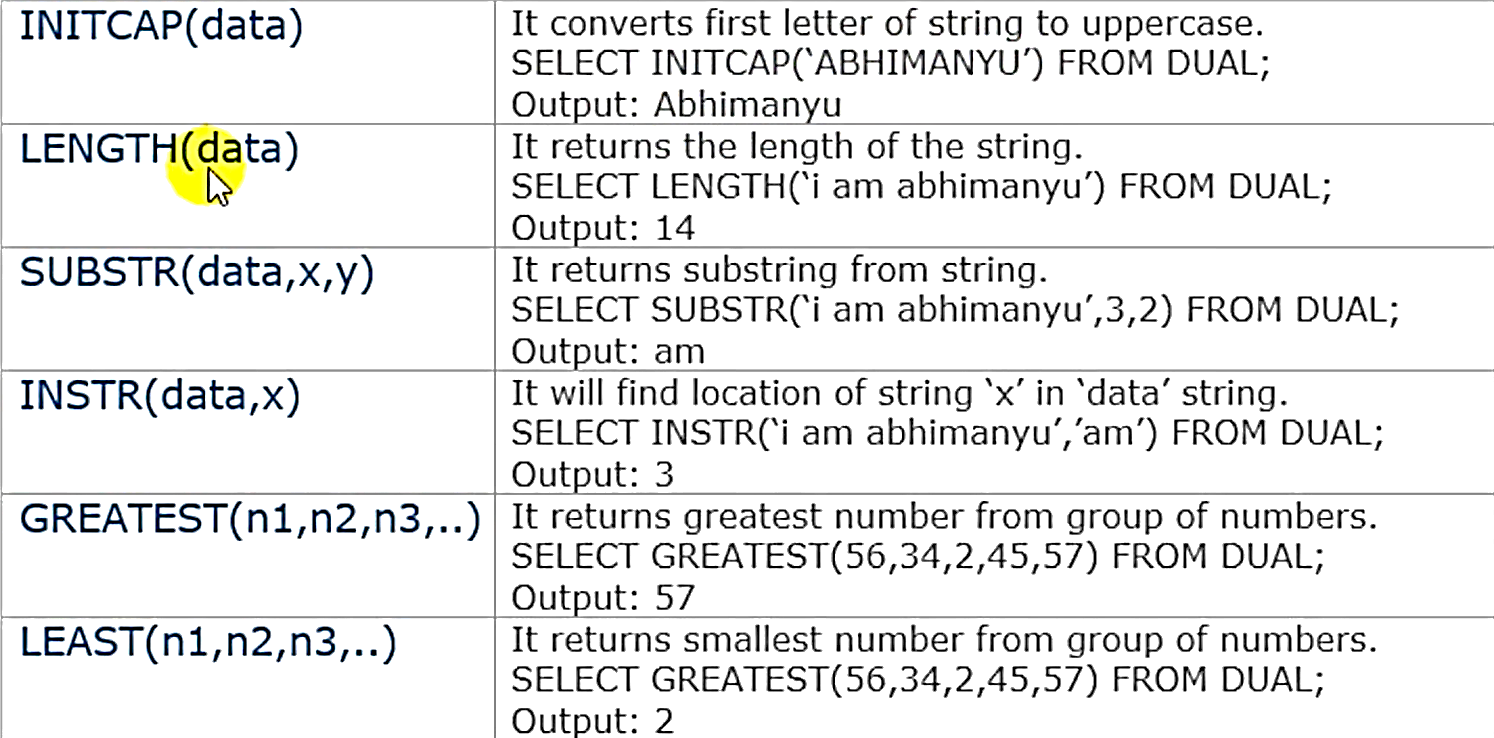
SQL> select stddev(salary) from empl;

1. **VARIANCE (X)**



SQL> select variance (salary) from empl;

CHARACTER FUNCTIONS IN ORACLE



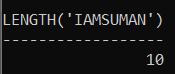
. Here White Spaces are also counted



**Example: -**

**INITCAP**

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

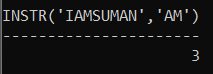
**LENGTH**

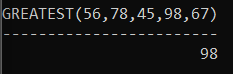
SQL> select length ('i am suman') from dual;

**SUBSTR**

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

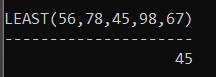
**INSTR**

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

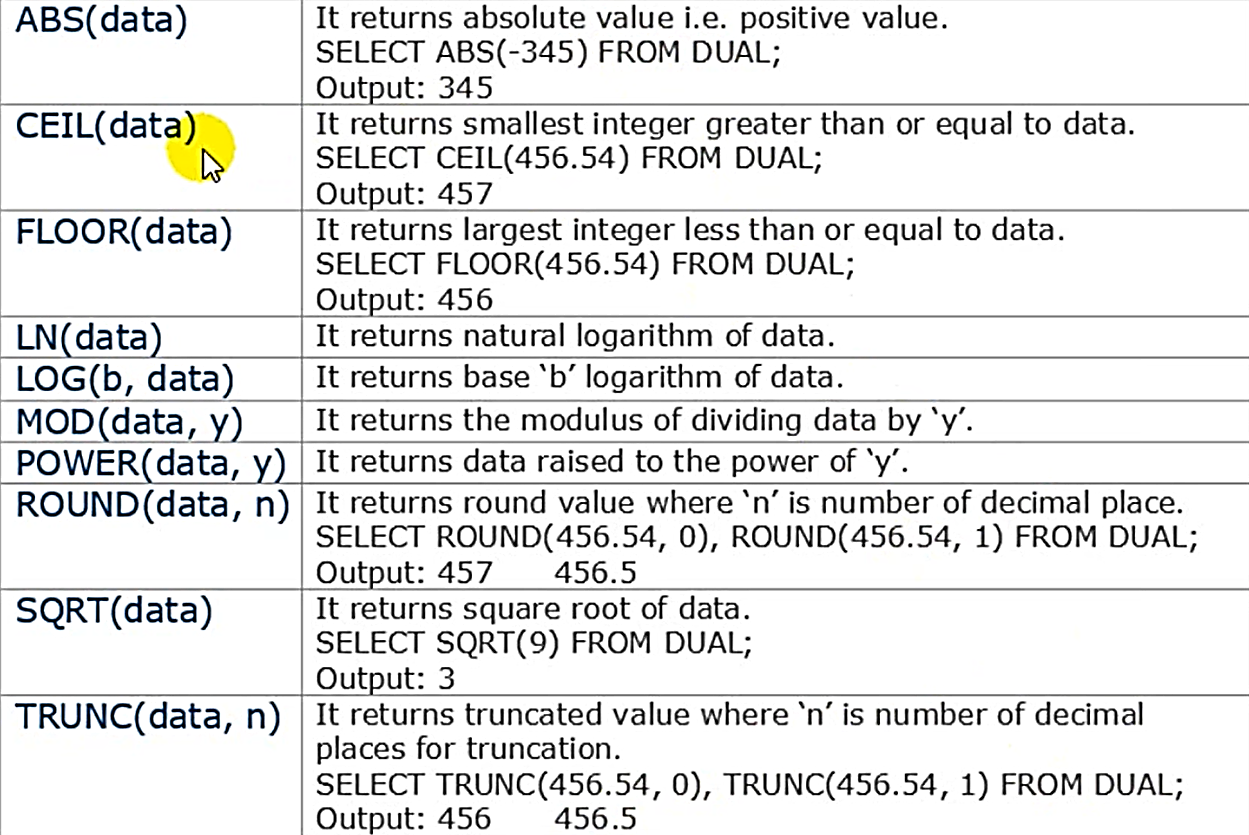
**GREATEST**

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

**LEAST**

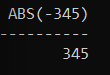
SQL> select least(56,78,45,98,67) from dual;

NUMBER FUNCTIONS IN ORACLE

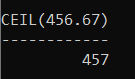


**Example: -**

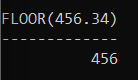
**ABS**

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

**CEIL**

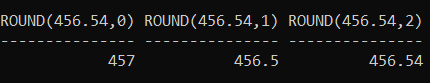
SQL> select ceil (456.67) from dual;

**FLOOR**

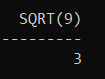
****SQL> select floor (456.34) from dual;

**ROUND**

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

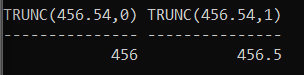


**SQRT**

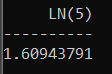
****SQL> select sqrt (9) from dual;

**TRUNC**

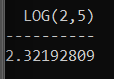
SQL> select trunc (456.54 ,0), trunc (456.54 ,1) from dual;



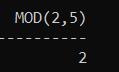
**LN**

SQL> select ln (5) from dual;

**LOG**

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

**MOD**

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

CONVERSION FUNCTIONS IN ORACLE

**TO\_DATE**

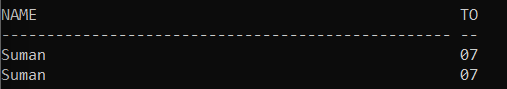
This is used to convert any string format into date format.

SQL> insert into birth values ('Suman', to\_date('2003-march-21', 'yyyy-mm-dd' ));

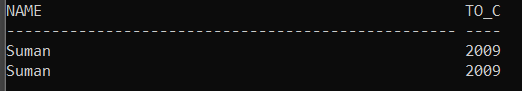
**TO\_CHAR**

If we want to see only specific part.

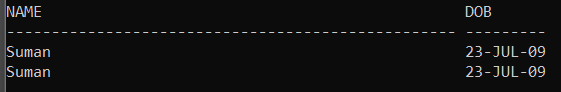
SQL> select name,to\_char (dob , 'mm') from birth;



SQL> select name,to\_char (dob , 'yyyy') from birth;



SQL> select name,dob from birth where to\_char (dob,'q')='3';



(because July is in quarter 3)

Quarter Division of Months

Q1 – January, February, March

Q2 – April, May, June

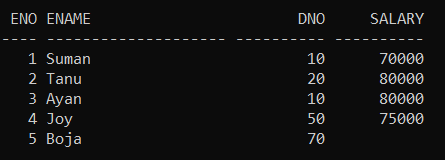
Q3 – July, August, September

Q4 – October, November, December

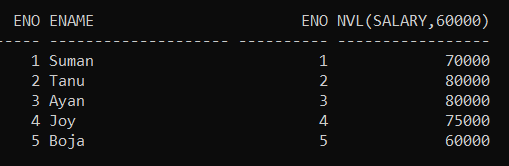
NVL

It fills default value in the null fields.

SQL> Select *fieldname, fieldname* nvl(*fieldname* , *default\_value*) from *tablename*;

**Example: -**

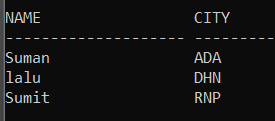
SQL> select eno,ename, eno, nvl(salary , 60000) from empl;



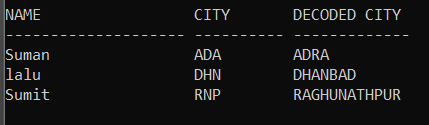
DECODE

Decode Function is used to expand a small abbreviation.

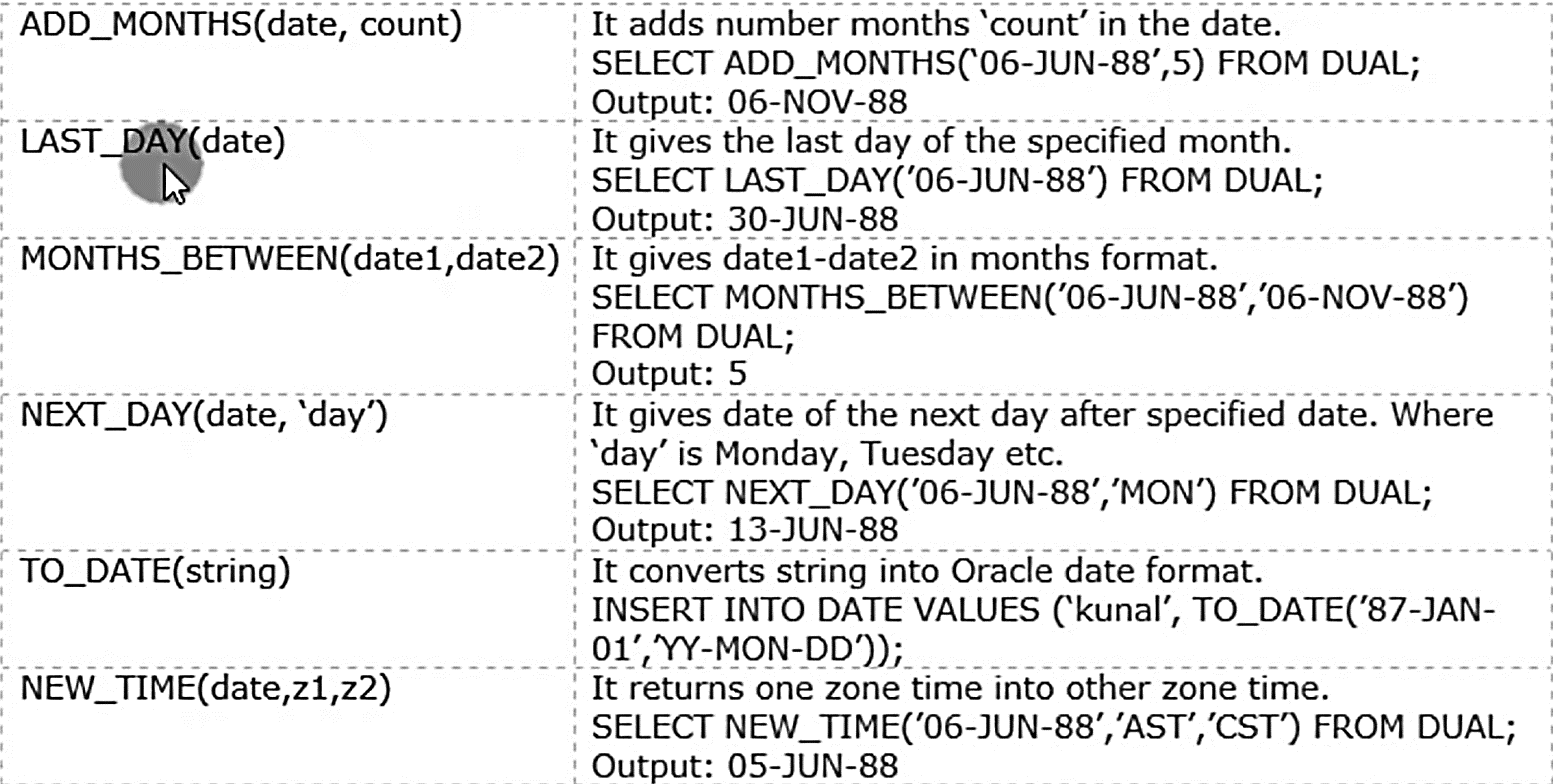
SQL> select *field*, decode (*field\_to\_be\_decoded*, '*small\_name*', '*full\_name*','NOT SPECIFIED')"*new\_column\_name*" from *tablename*;

**Example: -**

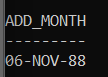
SQL> select name, city, decode (city, 'ADA', 'ADRA' , 'DHN' , 'DHANBAD' , 'RNP' , 'RAGHUNATHPUR','NOT SPECIFIED')"DECODED CITY" from city;



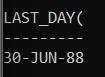
DATE FUNCTIONS



**Example: -**

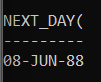
**ADD\_MONTHS**

SQL> select add\_months('6-jun-88',5) from dual;

**LAST\_DAY**

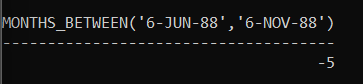
SQL> select last\_day('6-jun-88') from dual;

**NEXT\_DAY**

SQL> select next\_day('6-jun-88','WED') from dual;

**MONTHS\_BETWEEN**

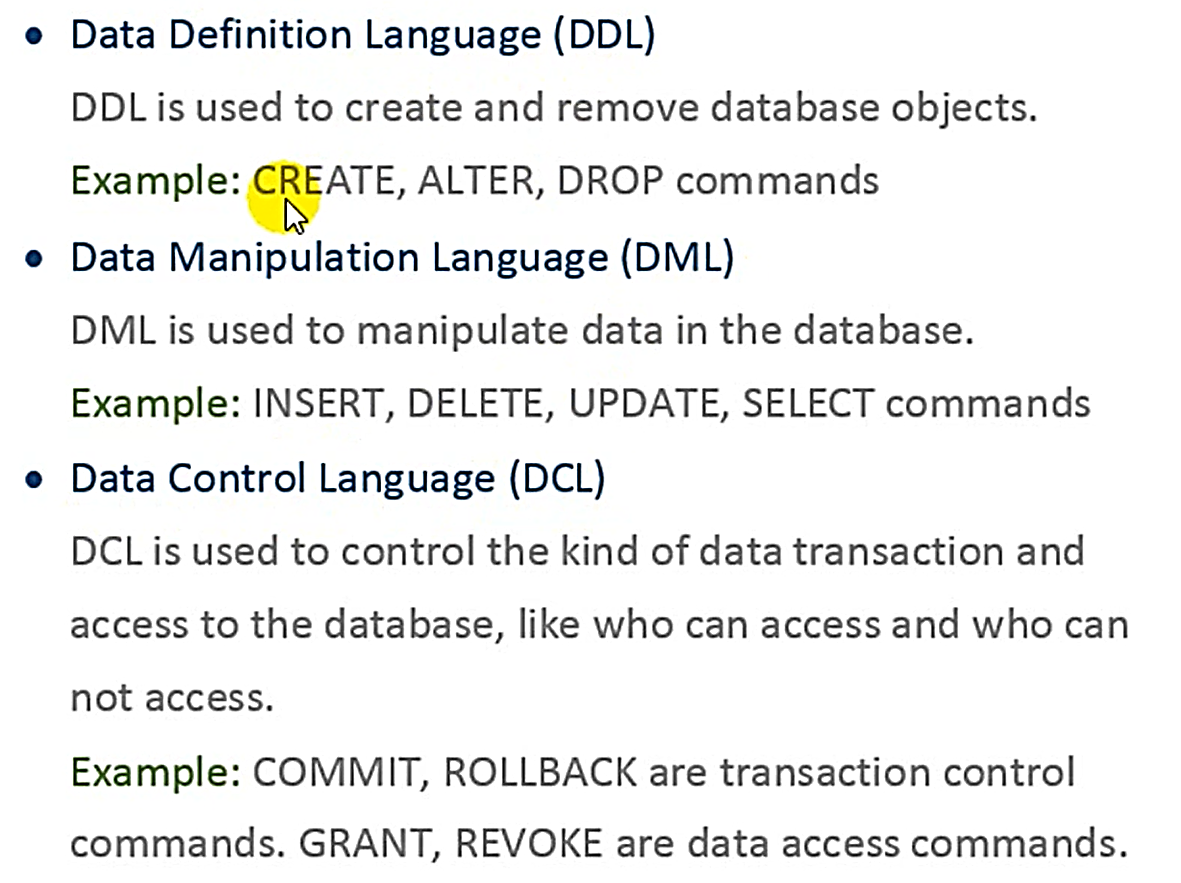
SQL> select months\_between ('6-jun-88','6-nov-88') from dual;



**NEW\_TIME**

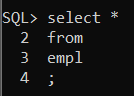
SQL> select new\_time('6-jun-88','gmt','bst') from dual;

SQL COMMAND TYPES IN ORACLE

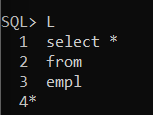


SQL BUFFER COMMANDS

Buffer Stores the Number of Lines in a query.



**TO SEE THE LINES**

SQL> L;

**TO SELECT A LINE AS THE NEWLINE**

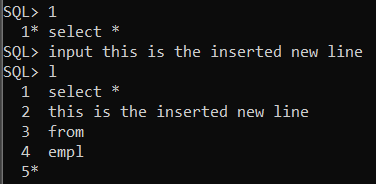
SQL> *line\_number;*

**Example: -**

****SQL> 1 **(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

**DELETE A LINE**

Select the Line and

SQL> del

**Example: -**

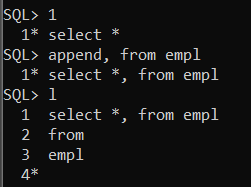
SQL> 2 **(Selects 2 as the new line)**

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

**APPEND IN A LINE**

Select the line and

SQL> Append, *append\_string*



**Example: -**

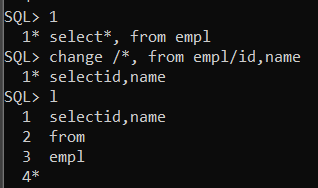
SQL> 1 **(Selects 1 as the new line)**

SQL> Append, from empl

**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

**RUN THE BUFFER COMMAND**

SQL> /

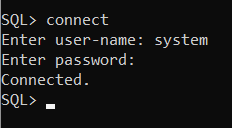
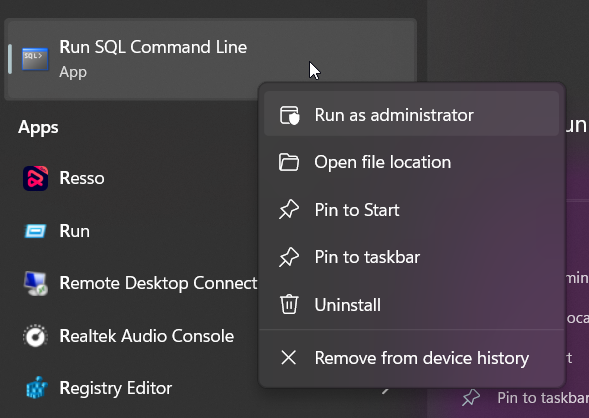
**ED OR EDIT BUFFER**

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

Now, SQL> ed

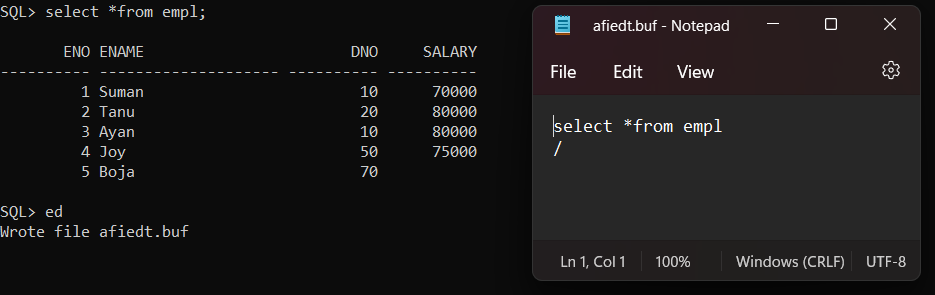
(It will create a buffer notepad file …sekhane amar run kora query ta lekha thakbe…ami notepad e kichu change korle seta command line e execute hobe…command execute korar jonno ‘/’ use korte hobe)

**Example: -**

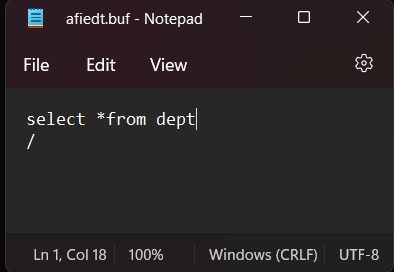


STEP 2 – Connect

STEP 1 – Run as Administrator



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



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

STEP 4 – Change in the notepad file

NOT NULL CONSTRAINT

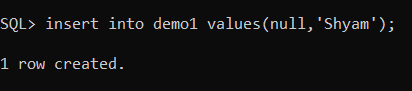
If we create a field as not null then we can’t insert null value in that. We can make multiple columns as not null.

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

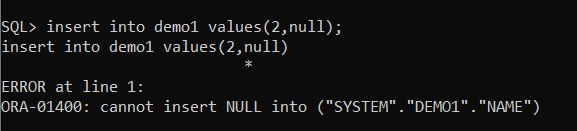
**Example: -**

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

**[Here the name field is not null]**



We can store null value in ID



But we can’t store null value in name iID

UNIQUE CONSTRAINTS

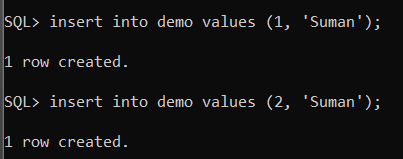
If we make a column as unique then we have to store unique values for each field in that column. We can make multiple columns unique.

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

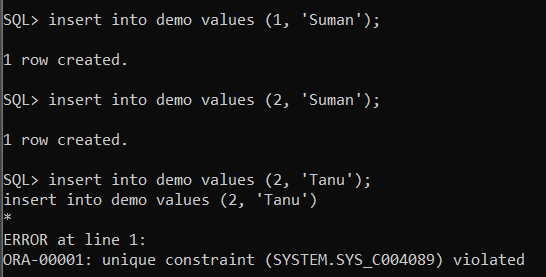
**Example: -**

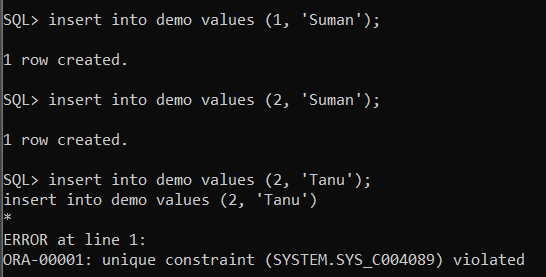
SQL> create table demo (id number unique, name varchar2(15));

**[Here the ID field is unique]**



The Name field can store duplicate values





The ID field can’t store duplicate values

PRIMARY KEY CONSTRAINT

Primary key identifies each record of the column uniquely.

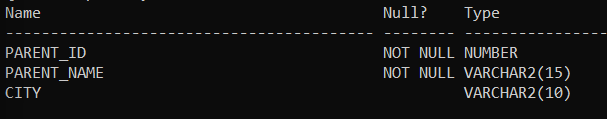
A Primary Key –

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

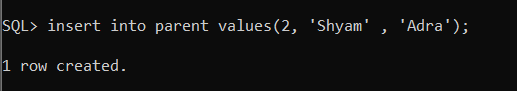
SQL> Create Table *tablename* (*fieldname datatype* primary key);

**Example: -**

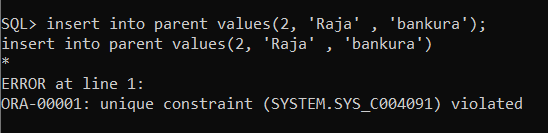
SQL> create table parent (parent\_id number primary key, parent\_name varchar2(15) not null, city varchar2(10));



Primary Key is Not Null



Unique value can be inserted



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 –

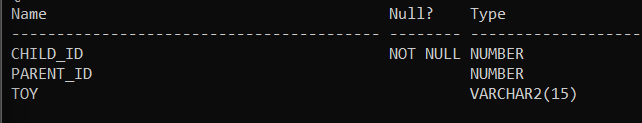
**(Method-1)**

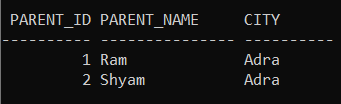
SQL> Create Table *tablename* (*fieldname (same as parent table primary key)* references *parent\_table\_name*);

**Example: -**

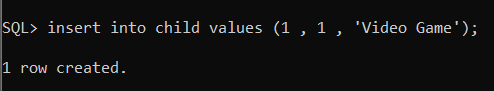
SQL> create table child (child\_id number primary key, parent\_id number references parent, toy varchar2(15));

**[ The ‘parent’ table is discussed in the primary key discussion section]**

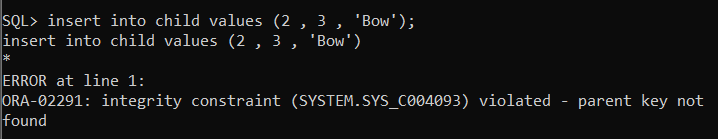




This is the Parent Table



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



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)*;

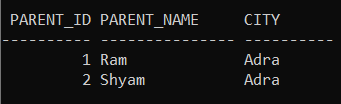
**We can Drop the Constraint to remove the Foreign Key using**

SQL> alter table *child\_table\_name* drop constraint *constraint\_name*;

**Example: -**

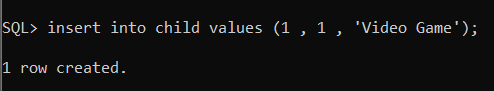
SQL> create table child(child\_id number primary key, parent\_id number, toy varchar2(15));

**[ Normally Creating a Table ]**

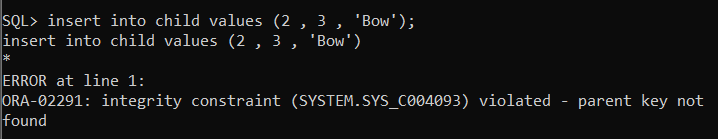
****

This is the Parent Table

SQL> alter table child add constraint fk\_parentchild foreign key(parent\_id) references parent(parent\_id);



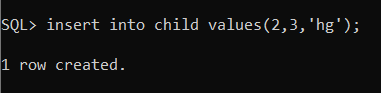
We can insert the values of ‘parent\_id’ which are available in the parent table (Ex:- 1)



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;



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.

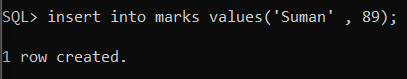
**(Check Constraint in Create Table)**

SQL> Create Table *tablename* (*fieldname datatype*, constraint *constraint\_name* check *(condition)*);

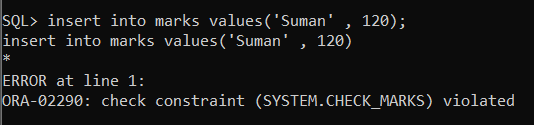
**Example: -**

**(1)**

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



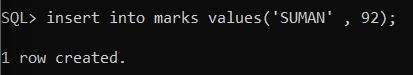
We can insert the value of marks which are between 0 and 100



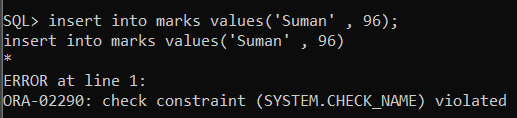
We can’t insert the value of marks which are not between 0 and 100

**(2)**

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



We can insert name in Uppercase



We cannot insert name without Uppercase

**(Check Constraint in Alter Table)**

Normally Create a Table

Now,

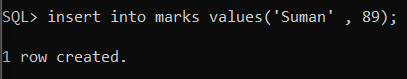
SQL> alter table *tablename* add constraint *constraint\_name* check *(condition)*

**Example: -**

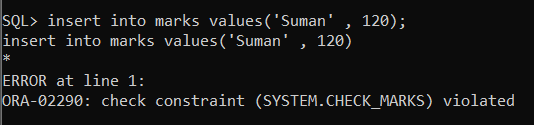
SQL> create table marks(name varchar2(15), marks number);

**[ Normally Creating a Table ]**

SQL> alter table marks add constraint check\_marks check (marks between 0 and 100);



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



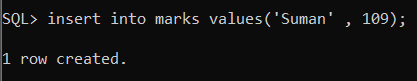
We can’t insert the value of marks which are not between 0 and 100

**Temporarily Disable the Constraint: -**

SQL> alter table *tablename* disable constraint *constraint\_name*;

**Example: -**

SQL> alter table marks disable constraint check\_marks;



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: -**

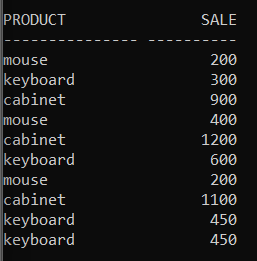
SQL> alter table *tablename* drop constraint *constraint\_name*;

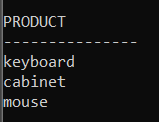
GROUP BY

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*.

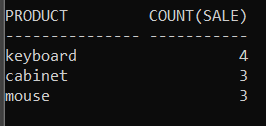
**Example: -**



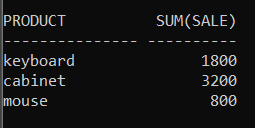
SQL> select product from orders group by product;

**[ It groups the products]**

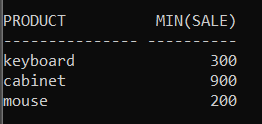
SQL> select product,count(sale) from orders group by product;

**[ It counts the number of sale of a product ]**

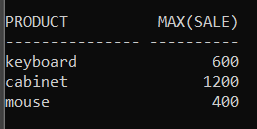
SQL> select product,sum(sale) from orders group by product;

**[ It counts the total sale of a product ]**

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

**[ It displays the minimum price of a product ]**

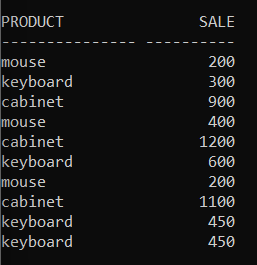
SQL> select product,max(sale) from orders group by product;

**[ It displays the maximum price of a product ]**

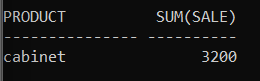
HAVING CLAUSE

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*.

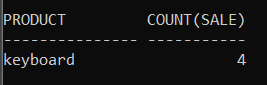
**Example: -**

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



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

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



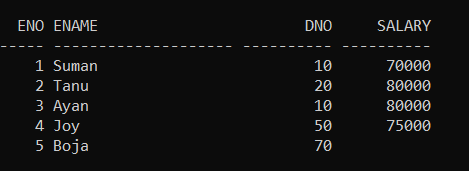
**[The product which has a sale of greater or equal than 4 times will be selected**

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.

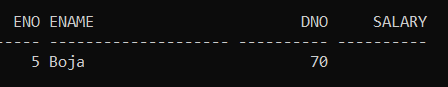
**Example: -**

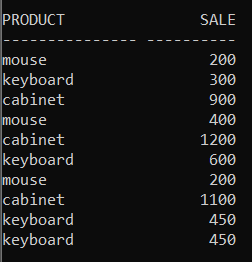


SQL> select \*from empl where eno = (select max(eno) from empl);

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

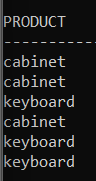
Now the query returns all the details of the employee whose eno is 5.



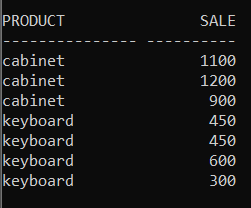
1. 

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,

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

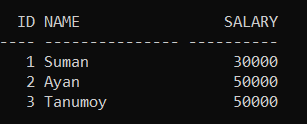
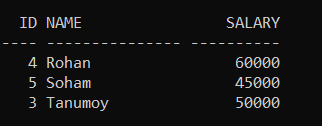


UNION OPERATOR

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

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

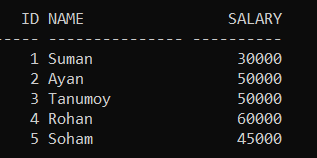
**Example: -**



emp2014

emp2013

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



It removed the

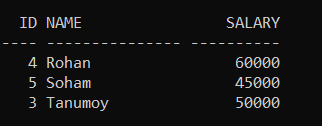
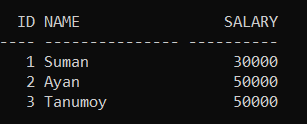
Duplicate value

UNION ALL

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

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

**Example: -**

****

emp2013

emp2014

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



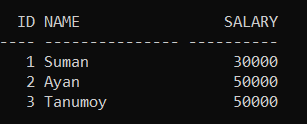
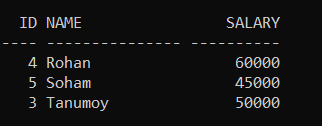
It does not remove the duplicates.

INSTERSECT

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

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

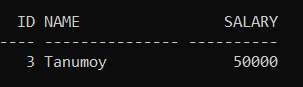
**Example: -**



emp2014

emp2013

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

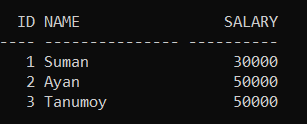
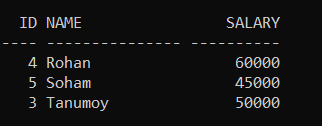


MINUS

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*;

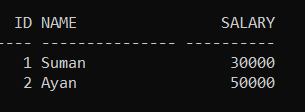
**Example: -**

****

emp2014

emp2013

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



***3 Tanumoy 50000***

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

EXECUTE COMMANDS WRITTEN IN PHYSICAL FILE

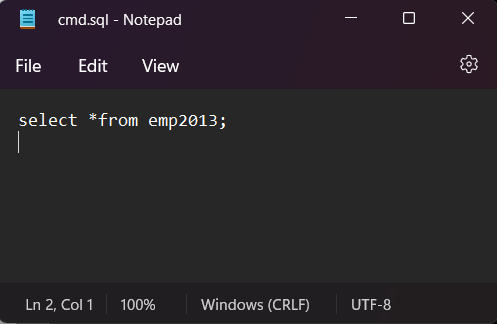
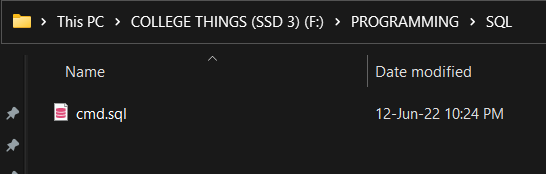
It runs the sql query which is present in any physical file in ‘.sql’ or ‘.txt’ format.

Make a .sql or .txt file in notepad.

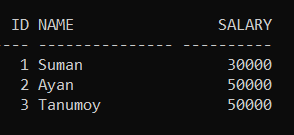
Now,

SQL> Start *file location (use ‘/’)* / *filename*;

**Example: -**

****

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



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.