Set 1

1-1.

Aggregate functions:

“COUNT()”, “SUM()” , “AVG()” , “ MAX()” , “MIN()”

COUNT ():

The Count function returns the no. of rows returned by a query.

Syntax:

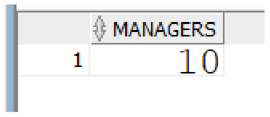
```
SELECT COUNT(Column_name)
FROM table_name
WHERE Condition;
```

Eg:- Display the total number of managers working in a company.

Query:

```
SELECT COUNT(DISTINCT MANAGER_ID) AS "MANAGERS" FROM
EMPLOYEES;
```

Output:



```
   MANAGERS
1        10
```

SUM():

The sum function add the column values in a query.

Syntax:

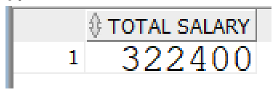
```
SELECT SUM(Column_name)
FROM table_name
WHERE condition;
```

Eg:- Display the total salary being paid to all employees.

Query:

```
SELECT SUM(SALARY) AS "TOTAL SALARY" FROM EMPLOYEES;
```

Output:



```
   TOTAL SALARY
1        322400
```

AVG():

AVG() Function is used to calculate average value of set of values.

Syntax:

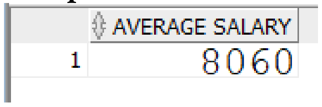
```
SELECT AVG(Column_name)
FROM table_name
WHERE condition;
```

Eg:- Display the average salary of employees

Query:

```
SELECT AVG(SALARY) AS "AVERAGE SALARY" FROM EMPLOYEES;
```

Output:



```
   AVERAGE SALARY
1            8060
```

MAX function:

This function is used to find min values from a set of values.

Syntax:

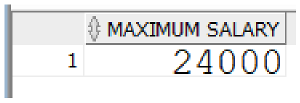
```
SELECT MAX(Column_name)
FROM table_name
WHERE condition;
```

Eg:- Display the maximum salary of employees

Query:

```
SELECT MAX(SALARY) AS "MAXIMUM SALARY" FROM EMPLOYEES;
```

Output:



```
   MAXIMUM SALARY
1           24000
```

MIN Function:

This function is used to find min value from a set o values.

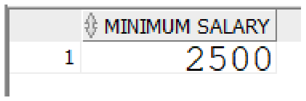
Syntax:

```
SELECT MIN(Colmun_name)

FROM table_name

WHER Condition;
```

Eg:- Display the minimum salary of employees
Query:

```
SELECT MIN(SALARY) AS "MINIMUM SALARY" FROM EMPLOYEES;
```

Output:



```
    MINIMUM SALARY
1             2500
```

1-2.

Write a PL/SQL code to calculate total average and class a student for marks in three Subjects

Code:

```
Declare

        snonumber(3) := 12;

        sname varchar2(20) := 'Suriya';

        m1 number(3) := 45;

        m2 number(3) := 67;

        m3 number(3) := 40;

        total number(4);

        avg1 number(5,2);

        class varchar(30);

begin

        total:=m1+m2+m3;

        avg1 := total/3;

        if m1>35 and m2>35 and m3>35 then

         if avg1 >= 60 then

                class :='First Class';

 end if;
```

```
                if avg1 >= 50 and avg1<60 then
                        class :='Second Class';
        end if;
                if avg1 > 40 and avg1<50 then
                        class :='Third Class';
         end if;
         else
                        class := 'Fail';
          end if;
          dbms_output.put_line('Name : '||sname);
          dbms_output.put_line('Total marks: '||total);
          dbms_output.put_line('Average marks: '||avg1);
          dbms_output.put_line('Class : '||class);
        end;
```

Output:

```
Name : Suriya
Total marks: 152
Average marks: 56.67
Class : Second Class
```

<u>Set 2</u>

2-1.

<u>GROUP BY:</u>

Group by clause is used to combine rows into groups based on matching values in specified columns.

- ❖ Group by clause is often used with aggregate functions.

Syntax:

```
SELECT Column_name(s), aggregate (Column_name)
FROM table_name(s)
GROUP by column_name;
```

Retrive average salary of each department

Query: SELECT D.DEPARTMENT_NAME, ROUND(AVG(E.SALARY),3) AS "AVERAGE SALARY" FROM EMPLOYEES E, DEPARTMENTS D
WHERE e.department_id=d.department_id
GROUP BY D.DEPARTMENT_NAME;

O/P:

```
Executive          19333.333
IT        5760
Finance 8600
Purchasing         4150
Shipping           5885.714
Sales     9616.667
Administration     4400
Marketing          9500
Human Resources            6500
Public Relations 10000
Accounting         10150
```

<u>HAVING:</u>

Having Clause is used with group by clause to specify conditions.

Syntax:

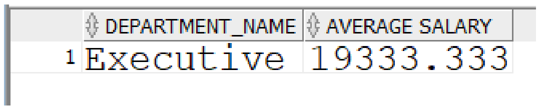
```
SELECT Column_name(s), aggregate function(column_name)
FROM table_names
GROUP by column_name
HAVING Conditions;
```

Eg:- Display the department name with average salary greater than 15,000.

Query:

```
SELECT   D.DEPARTMENT_NAME,  ROUND(AVG(E.SALARY),3)  AS  "AVERAGE
SALARY"   FROM   EMPLOYEES E, DEPARTMENTS  D

WHERE e.department_id=d.department_id

GROUP  BY  D.DEPARTMENT_NAME

HAVING  AVG(E.SALARY)>15000

ORDER BY D.DEPARTMENT_NAME;
```

Output:



```
   DEPARTMENT_NAME  AVERAGE SALARY
 1 Executive        19333.333
```

ORDER BY:

Order by clause is used to display the records in a ascending or descending order.

Syntax:

```
SELECT Column_name(s)

FROM table_name(s)

ORDER by column_name;
```

Retrieve first name of employees who are having salary greater than 20000 in ascending order.

Query: select first_name from employees where salary>20000 order by first_name;

Output:

```
first_name
  steven
```

2-2. Write a PL/SQL Program to find biggest of 3 numbers.

Code:

```
set serveroutput on

Declare

        a int := &a;

        b int := &b;
```

```
                c int := &c;
        Begin
                if(a>b) and (a>c) then
                        dbms_output.put_line('A is big');
                elsif(b>c) then
                        dbms_output.put_line('B is big');
                else
                        dbms_output.put_line('C is big');
                end if;
        end;
```

Output:

```
        Enter a: 5
        Enter b: 6
        Enter c: 7
        C is big
```

Set 3

3-1. **CONVERSION FUNCTIONS:**

TO_CHAR (): Converts a number or a date value to a VARCHAR2 character string with format model fmt.

Eg:

SELECT TO_CHAR(30000,’$99999’) FROM DUAL;

Output: $30000

SELECT TO_CHAR(SYSDATE,’dd,monyyyy’) FROM DUAL;

Output:11,jul 2017

TO_DATE (): Converts a character string representing a date to a date value according to the fmt specified (If fmt is omitted, format is DD-MONYY.)

Eg:

SELECT TO_DATE(‘25 JANUARY,17’) FROM DUAL;

Output: 25-JAN-17

SELECT TO_DATE(‘2 JANUARY,17’) FROM DUAL;

Output: 02-JAN-17

TO_NUMBER (): Converts a character string containing digits to a number with the optional format model fmt.

Eg:

SELECT TO_NUMBER(‘1210.72’,’9999.99’) FROM DUAL;

Output: 1210.72

3-2. **Write a PL/SQL Program to decode the Student grade**

Code:

```
set serveroutput on
declare
   grade char(1);
begin
grade := 'a';
   case
          when grade = 'a' then
         dbms_output.put_line('excellent');
      when grade = 'b' then
         dbms_output.put_line('Very Good');
      when grade = 'c' then
         dbms_output.put_line('Good');
      when grade = 'd' then
         dbms_output.put_line('Fair');
      when grade = 'f' then
         dbms_output.put_line('Poor');
      else
```

```
                    dbms_output.put_line('No Such Grade');
                 end case;
              end;
```

Output:

Output-1: excellent

Output-2: grade := ‘b’

Very good

Set 4

4-1.

STRING FUNCTIONS:

INITCAP ( ): This function returns the string with first letter of each word in uppercase.

Syntax: INITCAP (string1)

Eg: SELECT INITCAP(‘andhra pradesh’) FROM DUAL;

Output: Andhra Pradesh

LOWER ( ): This function returns the string in lower case.

Syntax: LOWER (string1)

Eg: SELECT LOWER(‘THE PEN IS MIGHTIER THAN THE SWORD’) FROM DUAL;

Output: the pen is mightier than the sword

UPPER ( ): This function returns the string in upper case.

Syntax: UPPER (string1)

Eg: SELECT UPPER(‘the pen is mightier than the sword’) FROM DUAL;

Output: THE PEN IS MIGHTIER THAN THE SWORD

CONCAT ( ): This function returns a string by appending string1 with string2.

Syntax:	CONCAT (string1, string2)

Eg: SELECT CONCAT('hello','every one') FROM DUAL;

Output: helloevery one

LENGTH ( ): This function gives length of the given string.

Syntax:	LENGTH (string)

Eg:SELECT LENGTH('Fortune favors the bold') FROM DUAL;

Output: 23

TRANSLATE ( ): This function returns a string after replacing some set of characters into another set.

Syntax:	TRANSLATE (MAIN STRING, FROM_STRING, TO_STRING)

Eg: SELECT TRANSLATE('delhi is the capital of india','i','a') FROM DUAL;

Output:delha as the capatal of andaa

LPAD ( ): This function returns a string as output after padding string2 to the left side of string1 to n length.

Syntax:	LPAD (STRING1, N, STRING2)

Eg: SELECT LPAD('india',20,'$') FROM DUAL;

Output: $$$$$$$$$$$$$$$india

RPAD ( ): This function returns a string as output after padding string2 to the right side of string1 to n length.

Syntax:	RPAD (STRING1, N, STRING2)

Eg: SELECT RPAD('india',20,'&') FROM DUAL;

Output:india&&&&&&&&&&&&&&&

4-2. **Write PL/SQL Program to find multiplication table for a given ‘n’ value.**

Code:

```
declare
          n int:= 5;
          res int := 0;
          i int := 1;
begin
          loop
                    res :=(n*i);
                     dbms_output.put_line(n||'*'||i||'='||res);
                    exit when i=10;
                    i:= i+1;
      end loop;
end;
```

Output:

```
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

<u>Set 5</u>

5-1. **DATE FUNCTIONS:**

SYSDATE: This function returns current date of system.

SELECT SYSDATE FROM DUAL;

Output: 21-MAY-21

ADD_MONTHS (): This function returns date d plus n months, i.e adds n months to the given date d.

Syntax: ADD_MONTHS (DATE, NO_OF_MONTHS)

SELECT ADD_MONTHS('15-AUG-1947',12) FROM DUAL;

Output:15-AUG-48

SELECT ADD_MONTHS('01-MAY-2017',15) FROM DUAL;

Output: 01-AUG-18

MONTHS_BETWEEN (): This function returns difference between given two dates.

Syntax: MONTHS_BETWEEN (DATE1, DATE2)

SELECT MONTHS_BETWEEN('19-SEP-16','17-MAY-16') FROM DUAL;

Output: 4.06451613

SELECT MONTHS_BETWEEN('19-FEB-16','17-MAY-16') FROM DUAL;

Output: -2.9354839

NEXT_DAY (): This function returns the date of the next weekday from the date specified.

Syntax: NEXT_DAY (DATE, 'WEEKDAY')

SELECT NEXT_DAY('15-AUG-1947','SUN') FROM DUAL;

Output: 17-AUG-47

SELECT NEXT_DAY('25-JUL-17','SUN') FROM DUAL;

Output: 30-JUL-17

LAST_DAY (): This function returns the date of the last day of the month.

Syntax: LAST_DAY (DATE)

SELECT LAST_DAY(‘15-AUG-1947’) FROM DUAL;

Output:31-AUG-47

SELECT LAST_DAY(‘22-APR-2017’) FROM DUAL;

Output:30-APR-17

5-2. **Write a pl/SQL Program to print Fibonacci series upto n terms**

Code:

```
    declare
          n int:= 10;
          a int := 0;
          b int := 1;
          c int;
    begin
          dbms_output.put_line('Fibonacci series upto '|| n || 'terms is : ');
          dbms_output.put_line(a);
        dbms_output.put_line(b);
         for i in 3..n loop
              c := a+b;
              a := b;
              b := c;
              dbms_output.put_line(c);
        end loop;
    end;
```

Output:

```
1
2
3
5
8
```

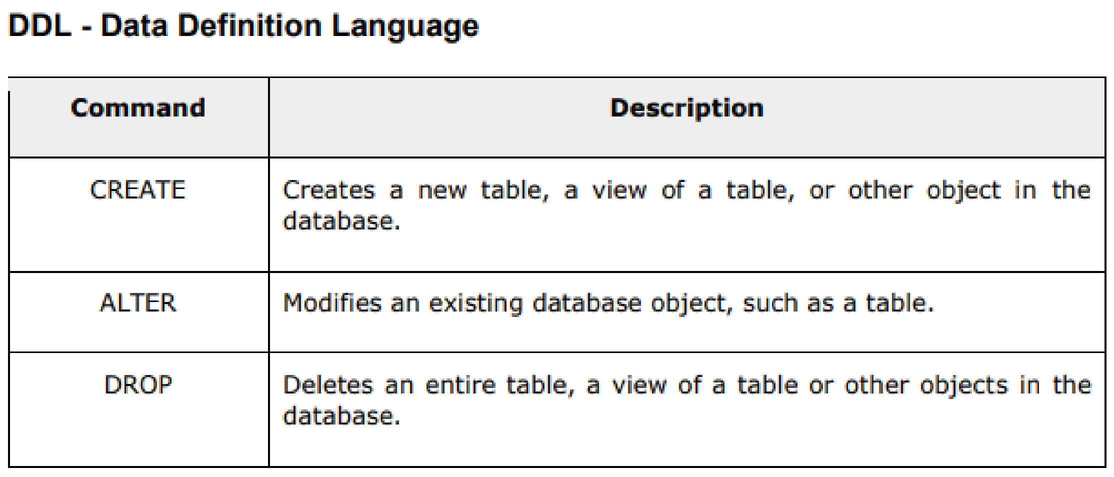
13

21

24

Set 6

6-1. DDL Commands:



DDL - Data Definition Language

Command	Description
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the database.

SYNTAX:

```
Create table table_name (“column1” “datatype”, “column2” “datatype”, “column3”
“datatype”, ….“column N” “datatype”);
```

```
        CREATE TABLE DEPARTMENT
        (
              DEPTCODE NUMBER(10),
              DeptName CHAR(30),
              LOCATION VARCHAR(33)
       );
```

Output:

```
Name      Null? Type
--------- ----- ------------
DEPTCODE        NUMBER(10)
DEPTNAME        CHAR(30)
LOCATION        VARCHAR2(33)
```

SYNTAX:

`ALTER table table_name add column1 datatype;`

EXAMPLE:

ALTER TABLE DEPARTMENT ADD PRIMARY KEY(DEPTCODE);

Output:

```
Name      Null?    Type
--------- -------- ------------
DEPTCODE  NOT NULL NUMBER(10)
DEPTNAME           CHAR(30)
LOCATION           VARCHAR2(33)
```

DROP TABLE:

- The drop table command deletes a table in the data base
- The following example SQL deletes the table “EMPLOYEE”

SYNTAX : **`DROP table table_name;`**

EXAMPLE: DROP table employee;

Dropping a table results in loss of all information stored in the table.

6-2. Write a PL/SQL Program to find the GCD of two numbers by using recursive procedure.

Source Code:

```
create or replace procedure gcd(x in out number, y in out number)
is
dif number;
begin
  if x<y then
```

```
            x:=x+y;
            y := x-y;
            x := x-y;
        end if;
        if x=1 or y=1 then
          dbms_output.put_line('1');
        elsif mod(x,y)=0 then
          dbms_output.put_line(y);
        else
            dif:=x-y;
            gcd(dif, y);
        end if;
      end;
```

<u>Executing the Procedure:</u>

```
      Declare
            a number;
            b number;
      begin
            a := 18;
            b:= 6;
            dbms_output.put_line(‘GCD of ‘||a||’and ‘||b||’ is’);
            gcd(a,b);
      end;
```

Output:

```
GCD of 18 and 6 is 6
```

<u>Set 7</u>

7-1.

<u>Different DML Operations (insert, delete, update):</u>

DML-Data Manipulation Language.

Data Manipulation Commands are used to manipulate data to the database.
Some of the data manipulation commands are

1. Insert
2. Update
3. Delete

Insert:

SQL insert statement is a sql query. It is used to insert a single multiple records in a table.

Syntax:

```
Insert into table name values (value 1, value 2, value 3);
```

insert into student values(„alekhya‟,501,‟hyderabad‟);

insert into student values(„deepti‟,502,‟guntur‟);

insert into student values(“ramya”,503,‟nellore‟);

The following table will be as follows:

NAME	ID	CITY
Alekhya	501	Hyderabad
Deepti	502	Guntur
Ramya	503	Nellore

Update:

- The SQL Commands update are used to modify the data that is already in the database.
- SQL Update statement is used to change the data of records held by tables which rows is to be update, it is decided by condition to specify condition, we use “WHERE” clause.
  - The update statement can be written in following form:

    Syntax:
    Update table_name set column_name=expression where condition;

Example:

```
Update students set name=‟rasi‟ where id=503
```

After update the table is as follows:

NAME	ID	CITY
Alekhya	501	Hyderabad
Deepti	502	Guntur
Rasi	503	Nellore

Delete:

- The SQL delete statement is used to delete rows from a table.
- Generally, delete statement removes one or more records from a table.

Syntax: `delete from table_name [where condition];`

Example: `Delete from students where id=501;`

O/P:

NAME	ID	CITY
Deepti	502	Guntur
Rasi	503	Nellore

7-2. Write a PL/SQL Program to find out the reverse of a given number.

Code:

```
declare
        n int:= 123;
        rem int;
        rev int := 0;
begin
  while n!=0 loop
      rem := mod(n, 10);
      rev := rem+(rev*10);
      n := trunc(n/10);
    end loop;
    dbms_output.put_line('Reverse number is: '||rev);
end;
```

Output:

```
Reverse number is : 321
```

<u>**Set 8**</u>

8-1.

SQL Querying with Where Clause

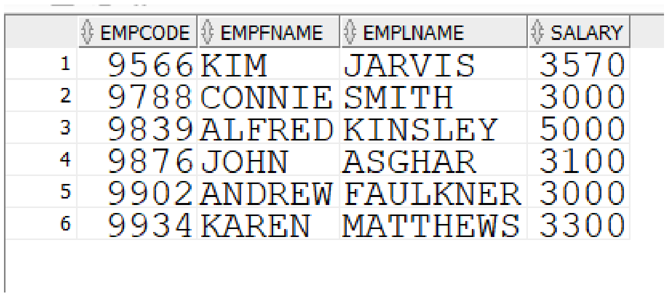
Maximum salary in IT department
(where)
select max(e.salary) from employees e, departments d
where d.department_id = e.department_id
and d.department_name='IT';
O/P: 9000

Minimum salary in sales department
(where)
select min(e.salary) from employees e, departments d
where d.department_id = e.department_id
and d.department_name='Sales';
O/P: 6200

Retrieve first names of employees whose salary is >20000
Query: select first_name from employees where salary>20000 order by first_name;
O/P: **Steven**

SELECT EMPCODE,EMPFNAME,EMPLNAME,SALARY FROM EMPLOYEE WHERE SALARY>=3000;

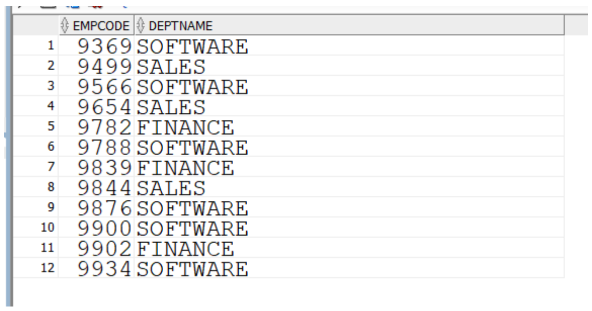
Output:



	EMPCODE	EMPFNAME	EMPLNAME	SALARY
1	9566	KIM	JARVIS	3570
2	9788	CONNIE	SMITH	3000
3	9839	ALFRED	KINSLEY	5000
4	9876	JOHN	ASGHAR	3100
5	9902	ANDREW	FAULKNER	3000
6	9934	KAREN	MATTHEWS	3300

SELECT EMPCODE, DEPTNAME FROM EMPLOYEE,DEPARTMENT WHERE EMPLOYEE.DEPTCODE=DEPARTMENT.DEPTCODE;

Output:



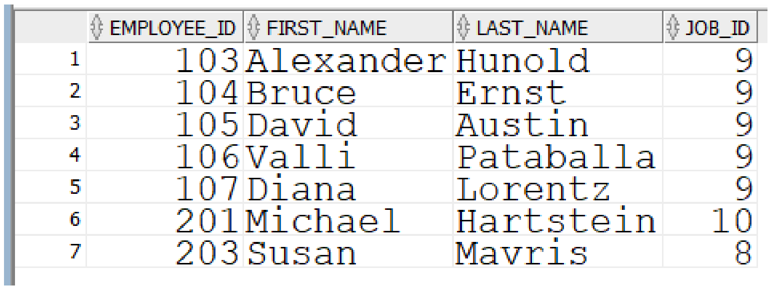
	EMPCODE	DEPTNAME
1	9369	SOFTWARE
2	9499	SALES
3	9566	SOFTWARE
4	9654	SALES
5	9782	FINANCE
6	9788	SOFTWARE
7	9839	FINANCE
8	9844	SALES
9	9876	SOFTWARE
10	9900	SOFTWARE
11	9902	FINANCE
12	9934	SOFTWARE

Display the employee details who are having job id 8,9 or 10.

Query:

```
SELECT  employee_id,      first_name,    last_name,     job_id FROM employees
WHERE   job_id IN (8, 9, 10);
```

Output:



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID
1	103	Alexander	Hunold	9
2	104	Bruce	Ernst	9
3	105	David	Austin	9
4	106	Valli	Pataballa	9
5	107	Diana	Lorentz	9
6	201	Michael	Hartstein	10
7	203	Susan	Mavris	8

8-2. Write a PL/SQL Program to find 1 to n prime number by using procedure.

Source Code:

```
create or replace procedure prime(n in number) is
flag number:=0;
begin
  dbms_output.put_line('Prime number from 1 to '||n);
  for i in 2..n loop
       for j in 1..trunc(i/2) loop
              if mod(i,j)=0 then
                     flag:=flag+1;
              end if;
        end loop;
       if flag=1 then
              dbms_output.put_line(i);
       end if;
       flag:=0;
  end loop;
end;
```

<u>Executing the procedures:</u>

1.Execute prime(25)

2.begin

```
      prime(25);
 end;
```

Output:

```
Prime number from 1 to 25
2
3
5
7
11
13
17
19
23
```

<u>Set 9</u>

9-1.

<u>IN:</u>In Keyword allow you to specify Multiple values in WHERE Clauses

Syntax :

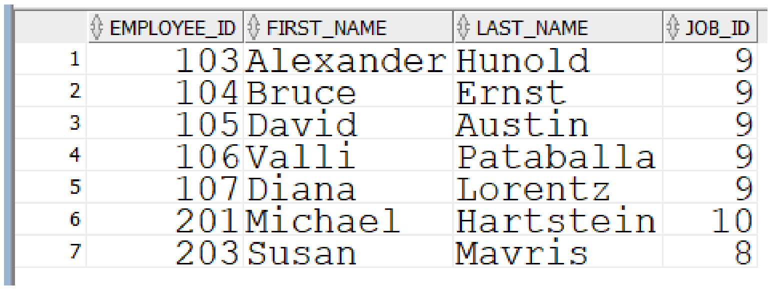
```
SELECT Column_name
FROM table_name
WHERE Column_name
IN(Value1, value2, ….);
```

Eg:- Display the employee details who are having job id 8,9 or 10.

Query:

```
SELECT    employee_id, first_name,   last_name,   job_id FROM employees
WHERE     job_id IN (8, 9, 10);
```

Output:



	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	JOB_ID
1	103	Alexander	Hunold	9
2	104	Bruce	Ernst	9
3	105	David	Austin	9
4	106	Valli	Pataballa	9
5	107	Diana	Lorentz	9
6	201	Michael	Hartstein	10
7	203	Susan	Mavris	8

ANY: It is a logical operator that compares a value with a set of values returned by a sub-query.

- The ANY Operator must be proceded by comparison Operator (< , > , <= , >= , <>).

Syntax:

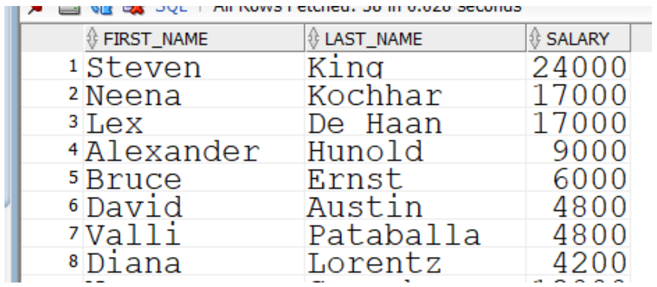
```
SELECT Cloumn_name(s)
FROM table_name(s)
WHERE Column_name   comparison  Operator
ANY
(Select column_name FROM Table_name
Where Condition);
```

Eg:- Display names of the employees whose salaries are not equal to the department number 4.

Query:

```
SELECT    first_name,  last_name,    salary
FROM    employees WHERE    salary <> ANY (SELECT
salary  FROM   employees  WHERE department_id = 4);
```

Output:



	FIRST_NAME	LAST_NAME	SALARY
1	Steven	King	24000
2	Neena	Kochhar	17000
3	Lex	De Haan	17000
4	Alexander	Hunold	9000
5	Bruce	Ernst	6000
6	David	Austin	4800
7	Valli	Pataballa	4800
8	Diana	Lorentz	4200

38 Records Found….

<u>ALL:</u> It is a logical operator that compares a single value that compares a single column.

- Set of values returned by a sub Queries.
- Here Condition is true , if it satisfy all values in a set

Syntax:

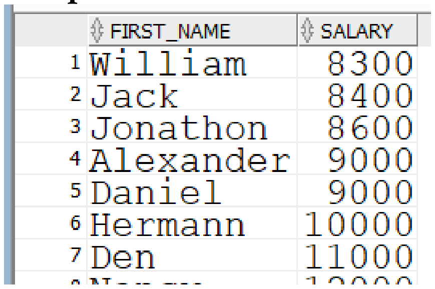
```
SELECT Column_name(s)

FROM table_name(s)

WHERE Column_name Comparison  Operator  ALL  (Select
Column_name From table_name where condition);
```

Eg:- Display the first name of all employees whose salary is greater than all employees in department 5.

Query:

```
SELECT first_name, salary FROM employees WHERE salary > all

(SELECT salary from employees where department_id=5);
```

Output:



```
   FIRST_NAME   SALARY
 1 William        8300
 2 Jack           8400
 3 Jonathon       8600
 4 Alexander      9000
 5 Daniel         9000
 6 Hermann       10000
 7 Den           11000
```

9-2. Write a PL\SQL program to handle built-in execution NO_DATA_FOUND

Program:

```
Set serveroutput on
Declare
        emp employees.employee_id%type:=500;
        empname employees.first_name%type;
begin
        select first_name into empname from employees where employee_id=emp;
        dbms_output.put_line('employee first name '||empname);
        exception when NO_DATA_FOUND then
                dbms_output.put_line('invalid employee id');
End;
```

Output:

1.Invalid employee id

2.Employee first name steven

Set 10

10-1.

UNION: The UNION Operator is used to combine the result set of two or more select statements.

- ❖ Every select statement within union must have same number of column in same order.
- ❖ It will allow distinct values.

Syntax:

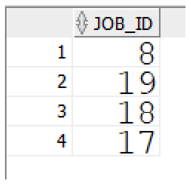
```
SELECT Column_name(s) FROM table_name
UNION
SELECT Column_name(s) FROM table_name;
```

Eg:- Display the job id's who works either department number 4 or 5

Query:

```
Select job_id FROM employees where department_id=4

UNION

SELECT job_id FROM employee where department-id=5;
```

Output:



```
   JOB_ID
1       8
2      19
3      18
4      17
```

INTERSECT: INTERSECT Operator is used t return distinct rows of two or more results sets from a select statements.

Syntax:

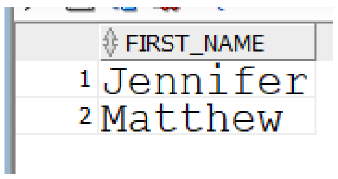
```
SELECT Column_name(s) FROM table_name;

INTERSECT

SELECT Column_name(s) FROM table_name;
```

Eg:- Display the common first names from both employees and dependents.

Query:

```
SELECT first_name FROM employees

INTERSECT

SELECT first_name FROM dependents;
```

Output:



	FIRST_NAME
1	Jennifer
2	Matthew

Views:

View is a virtual table which is generated based on the result set of an SQL statement.

- A view is also have rows and column same as a table in a database.

Create a view on Sales department staff of data in the employees table

Query:

```
CREATE VIEW SALES_TABLE AS SELECT EMPLOYEE_ID, FIRST_NAME,
DEPARTMENT_ID

 FROM EMPLOYEES WHERE DEPARTMENT_ID = 8;
```

Output:

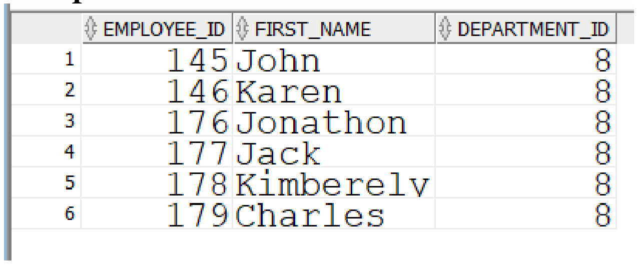
```
View SALES_TABLE created.
```

Display a view of sales_table.

Query:

```
SELECT * FROM SALES_TABLE;
```

Output:



	EMPLOYEE_ID	FIRST_NAME	DEPARTMENT_ID
1	145	John	8
2	146	Karen	8
3	176	Jonathon	8
4	177	Jack	8
5	178	Kimberelv	8
6	179	Charles	8

Drop a view of sales_table

Query:

```
DROP VIEW SALES_TABLE;
```

Output:

```
View SALES_TABLE dropped.
```

10-2. Write a PL\SQL program to swap numbers

Program:

```
 Set serveroutput  on
Declare
          a number:=5;
          b number:=10;
          temp number;
Begin
       dbms_output.put_line('before swapinga:'||a||'b:'||b);
       temp:=a;
       a:=b;
       b:=temp;
       dbms_output.put_line('after swaping a:'||a||'b:'||b);
End;
```

Output:

```
Before swaping a:5  b:10
After swaping a:10 b:5
```