EXPERIMENT: 1

Aim: Queries for Creating, Dropping, and Altering Tables and insert row into a table (use constraints while creating tables) examples using Select Command.

Procedure:

1. Creation of emp & dept table in SQL:

```
SQL>create table dept(deptno number(2,0) primary key, dname varchar2(14) NOT NULL,
loc varchar2(13) NOT NULL);
```

Table created.

```
SQL>create table emp(empno number(4,0) primary key,ename varchar2(10) NOT NULL,
job varchar2(9) NOT NULL, mgr number(4,0),hiredate date,sal number(7,2) NOT NULL,
comm number(7,2),deptno  number(2,0),foreign key (deptno) references dept (deptno));
```

Table created.

1. View Structure/schema of emp & dept table in SQL:

```
SQL> select *from emp;

no rows selected

SQL> select *from dept;
no rows selected


SQL> desc emp;
Name               Null?          Type

------------------------------------- -------- ------------------------------

EMPNO         NOT NULL    NUMBER(4)

ENAME                          VARCHAR2(10)

JOB                             VARCHAR2(9)

MGR                             NUMBER(4)

HIREDATE                        DATE

SAL                             NUMBER(7,2)

COMM                            NUMBER(7,2)

DEPTNO                          NUMBER(2)


SQL> desc dept;

Name                      Null?   Type

------------------------------------- -------- ---------------------------------------------

DEPTNO        NOT NULL     NUMBER(2)
```

```
DNAME                          VARCHAR2(14)
LOC                            VARCHAR2(13)
```

2. Insert the values in emp & dept table in SQL:

There are several ways to insert the values in the existing table

Query to insert single record in the existing table:

```
SQL> insert into dept(deptno,dname,loc) values(20,’admin’,’hyd’);
1 row created.
```

Query to insert multiple records in the existing table:

```
 SQL>insert into dept values(&deptno,'&dname','&loc');
 Enter value for deptno: 10
Enter value for dname: sales
 Enter value for loc: vijayawada
old 1: insert into dept values(&deptno,'&dname','&loc')
new 1: insert into dept values(10,'sales','vijayawada')
1 row created.
SQL>/
Enter value for deptno: 20
Enter value for dname: admin
Enter value for loc: hyd
old 1: insert into dept values(&deptno,'&dname','&loc')
1 row created.
SQL> /
Enter value for deptno: 30
Enter value for dname: marketing
Enter value for loc: vzg
old 1: insert into dept values(&deptno,'&dname','&loc')
new 1: insert into dept values(30,'marketing','vzg')
1 row created.
```

3. Select Command: this command is used to print the record from the existing table.

View all records in dept table:

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

```
DEPTNO      DNAME        LOC
--------  -----------  ----------
10  sales        vijayawada
20  admin        hyd
30  marketing  vzg
```

View records basing on given criteria on specific column.

1. View single column from existing table.

```
SQL>select dname from dept;
DNAME
--------------
Sales Admin Marketing
```

2. View specific record(s) from existing table based on given condition.

```
SQL> select *from dept where dname='sales';
  DEPTNO     DNAME        LOC
--------  -----------  ----------
              10  sales    Vijayawada
```

4. <u>Alter the columns in dept table</u>

1. To add a new column in the table

```
   SQL>alter table dept add (dsal number(5,0));

   Table Altered
```

```
SQL> desc dept;
 Name                    Null?          Type
 ----------------------------------- -------- ---------------------------------

 DEPTNO      NOT NULL            NUMBER(2)

 DNAME                          VARCHAR2(14)

 LOC                            VARCHAR2(13)
 DSAL                            NUMBER(5,0)
```

2. To modify existing column in the table

```
SQL>alter table dept modify (dsal number(6,1));

Table Altered
```

```
SQL> desc dept;
 Name                    Null?          Type
 ----------------------------------- -------- --------------------------------

 DEPTNO       NOT NULL           NUMBER(2)

 DNAME                           VARCHAR2(14)
 LOC                             VARCHAR2(13)
 DSAL                             NUMBER(6,1)
```

5. Drop the table in the database

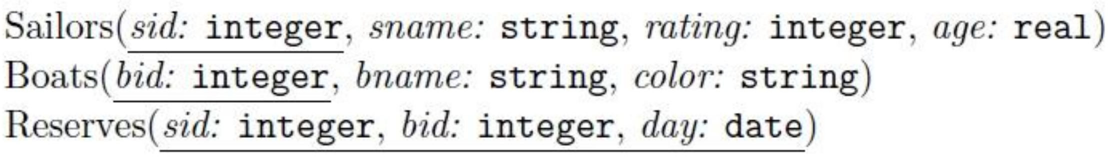
```
SQL>drop table emp;
Table dropped
```

```
SQL> desc emp;


Error: table or view does not exists
```

EXPERIMENT - 2

QUERIES (ALONG WITH SUB QUERIES) USING ANY, ALL, IN, EXISTS, NOT EXISTS, UNION, INTERSECT



Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

1. **Create a Table Sailors with sid, sname, rating and age.**

SQL> create table Sailors (sid number(3) primary key,sname varchar(15),rating int,age number(3,1));

Table created.

2. **Describe Sailors Table**

SQL> desc Sailors;

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(3)
 SNAME                                              VARCHAR2(15)
 RATING                                             NUMBER(38)
 AGE                                                NUMBER(3,1)
```

3. **Insert values into Sailors Table**

```
SQL> insert into Sailors values(&sid,'&sname',&rating,&age);
Enter value for sid: 22
Enter value for sname: Dustin
Enter value for rating: 7
Enter value for age: 45.0
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(22,'Dustin',7,45.0)

1 row created.
```

```
SQL> /
Enter value for sid: 29
Enter value for sname: Brutus
Enter value for rating: 1
Enter value for age: 33.0
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(29,'Brutus',1,33.0)

1 row created.

SQL> /
Enter value for sid: 31
Enter value for sname: Lubber
Enter value for rating: 8
Enter value for age: 55.5
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(31,'Lubber',8,55.5)

1 row created.

SQL> /
Enter value for sid: 32
Enter value for sname: Andy
Enter value for rating: 8
Enter value for age: 25.5
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(32,'Andy',8,25.5)

1 row created.

SQL> /
Enter value for sid: 58
Enter value for sname: Rusty
Enter value for rating: 10
Enter value for age: 35.0
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(58,'Rusty10',10,35.0)

1 row created.
```

```
SQL> /
Enter value for sid: 64
Enter value for sname: Horatio
Enter value for rating: 7
Enter value for age: 35.0
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(64,'Horatio',7,35.0)

1 row created.

SQL> /
Enter value for sid: 71
Enter value for sname: Zorba
Enter value for rating: 10
Enter value for age: 16.0
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(71,'Zorba',10,16.0)

1 row created.

SQL> /
Enter value for sid: 74
Enter value for sname: Horatio
Enter value for rating: 9
Enter value for age: 35.0
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(74,'Horatio',9,35.0)

1 row created.

SQL> /
Enter value for sid: 85
Enter value for sname: Art
Enter value for rating: 3
Enter value for age: 25.5
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(85,'Art',3,25.5)

1 row created.
```

```
SQL> /
Enter value for sid: 95
Enter value for sname: Bob
Enter value for rating: 3
Enter value for age: 63.5
old   1: insert into Sailors values(&sid,'&sname',&rating,&age)
new   1: insert into Sailors values(95,'Bob',3,63.5)

1 row created.
```

4. **Display values in the Sailors Table**

```
SQL> select * from Sailors;

       SID  SNAME            RATING       AGE
---------- -------------- ---------- ----------
        22  Dustin              7          45
        29  Brutus              1          33
        31  Lubber              8          55.5
        32  Andy                8          25.5
        58  Rusty              10          35
        64  Horatio             7          35
        71  Zorba              10          16
        74  Horatio             9          35
        85  Art                 3          25.5
        95  Bob                 3          63.5

10 rows selected.
```

1. **Create a Table Boats with bid, bname and color.**

```
SQL> create table Boats(bid number(3) primary key,bname varchar(10),color
varchar(10));


Table created.
```

2. **Describe Boats Table**

```
SQL> desc Boats
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 BID                                       NOT NULL NUMBER(3)
 BNAME                                              VARCHAR2(10)
 COLOR                                              VARCHAR2(10)
```

3. **Insert values into Boats Table**

```
SQL> insert into Boats values(&bid,'&bname','&color');
Enter value for bid: 101
Enter value for bname: Interlake
Enter value for color: blue
old   1: insert into Boats values(&bid,'&bname','&color')
new   1: insert into Boats values(101,'Interlake','blue')

1 row created.

SQL> /
Enter value for bid: 102
Enter value for bname: Interlake
Enter value for color: red
old   1: insert into Boats values(&bid,'&bname','&color')
new   1: insert into Boats values(102,'Interlake','red')
```

```
1 row created.

SQL> /
Enter value for bid: 103
Enter value for bname: Clipper
Enter value for color: green
old   1: insert into Boats values(&bid,'&bname','&color')
new   1: insert into Boats values(103,'Clipper','green')

1 row created.

SQL> /
Enter value for bid: 104
Enter value for bname: Marine
Enter value for color: red
old   1: insert into Boats values(&bid,'&bname','&color')
new   1: insert into Boats values(104,'Marine','red')

1 row created.
```

4. **Display values in the Boats Table**

```
SQL> select * from Boats;

       BID BNAME      COLOR
---------- ---------- ----------
       101 Interlake  blue
       102 Interlake  red
```

```
     103 Clipper    green
     104 Marine     red
```

1. **Create a Table Reserves with sid, bid and day.**

```
SQL> create table Reserves(sid number(3),bid number(3),day Date,primary key(sid,bid,day),foreign key(sid) references Sailors(sid),foreign key(bid) references Boats(bid));

Table created.
```

2. **Describe Reserves Table**

```
SQL> desc Reserves;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SID                                       NOT NULL NUMBER(3)
 BID                                       NOT NULL NUMBER(3)
 DAY                                       NOT NULL DATE
```

3. **Insert values into Reserves Table**

```
SQL> insert into Reserves values(&sid,&bid,'&day');
Enter value for sid: 22
Enter value for bid: 101
Enter value for day: 10-oct-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(22,101,'10-oct-1998')

1 row created.
```

```
SQL> /
Enter value for sid: 22
Enter value for bid: 102
Enter value for day: 10-oct-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(22,102,'10-oct-1998')

1 row created.

SQL> /
Enter value for sid: 22
Enter value for bid: 103
Enter value for day: 10-aug-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(22,103,'10-aug-1998')

1 row created.

SQL> /
Enter value for sid: 22
Enter value for bid: 104
Enter value for day: 10-jul-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(22,104,'10-jul-1998')

1 row created.
```

```
SQL> /
Enter value for sid: 31
Enter value for bid: 102
Enter value for day: 11-oct-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(31,102,'11-oct-1998')

1 row created.

SQL> /
Enter value for sid: 31
Enter value for bid: 103
Enter value for day: 11-jun-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(31,103,'11-jun-1998')

1 row created.

SQL> /
Enter value for sid: 31
Enter value for bid: 104
Enter value for day: 11-dec-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(31,104,'11-dec-1998')

1 row created.
```

```
SQL> /
Enter value for sid: 64
Enter value for bid: 101
Enter value for day: 09-may-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(64,101,'09-may-1998')

1 row created.

SQL> /
Enter value for sid: 64
Enter value for bid: 102
Enter value for day: 09-aug-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(64,102,'09-aug-1998')

1 row created.

SQL> /
Enter value for sid: 74
Enter value for bid: 103
Enter value for day: 09-aug-1998
old   1: insert into Reserves values(&sid,&bid,'&day')
new   1: insert into Reserves values(74,103,'09-aug-1998')

1 row created.
```

4. Display values in the Reserves Table

```
SQL> select * from Reserves;

       SID        BID DAY
---------- ---------- ---------
        22        101 10-OCT-98
        22        102 10-OCT-98
        22        103 10-AUG-98
        22        104 10-JUL-98
        31        102 11-OCT-98
        31        103 11-JUN-98
        31        104 11-DEC-98
        64        101 09-MAY-98
        64        102 09-AUG-98
        74        103 09-AUG-98

10 rows selected.
```

1) ANY

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

Syntax:

SELECT *column_name(s)* FROM *table_name* WHERE *column_name operator* ANY
(SELECT *column_name* FROM *table_name* WHERE *condition*);

1. **Find Sailors whose rating is better than some Sailors called Horatio.**

```
SQL> select S.sid from Sailors S where S.rating > any (select S2.rating from Sailors S2 where S2.sname='Horatio');


      SID
----------
       58
       71
       74
       31
       32
```

2) ALL

The ALL operator:

- returns a boolean value as a result
- returns TRUE if ALL of the subquery values meet the condition
- is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

Syntax:

SELECT *column_name(s)* FROM *table_name* WHERE *column_name operator* ALL
(SELECT *column_name* FROM *table_name* WHERE *condition*);

2. **Find the Sailors with the highest rating**

```
SQL> select s.sid from Sailors s where S.rating >= all (select S2.rating from
Sailors S2);


       SID
----------
        58
        71
```

3) IN

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

Syntax:

SELECT *column_name(s)* FROM *table_name*
WHERE *column_name* IN (*SELECT STATEMENT*);

3. Find the names of Sailors who have reserved boat 103.

```
SQL> select s.sname from Sailors s where s.sid in (select r.sid from Reserves r
where r.bid=103);


SNAME
---------------
Dustin
Lubber
Horatio
```

4. Find the names of Sailors who have reserved red boat

```
SQL> select s.sname from Sailors s where s.sid in (select r.sid from Reserves r
where r.bid in (select b.bid from Boats b where b.color='red'));
```

```
SNAME
---------------
Dustin
Lubber
Horatio
```

4) EXISTS

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

Syntax:

SELECT *column_name(s)* FROM *table_name* WHERE EXISTS
(SELECT *column_name* FROM *table_name* WHERE *condition*);

5. Find the names of Sailors who have reserved boat 103.

```
SQL> select s.sname from Sailors s where exists (select * from Reserves r
where r.bid = 103 and r.sid = s.sid);
```

```
SNAME
---------------
Dustin
Lubber
Horatio
```

5) NOT EXISTS

The NOT EXISTS operator is used to test for the NOT existence of any record in a subquery.

The NOT EXISTS operator returns TRUE OR FALSE if the subquery returns one or more records.

Syntax:

SELECT *column_name(s)* FROM *table_name* WHERE NOT EXISTS
(SELECT *column_name* FROM *table_name* WHERE *condition*);

6. **Find the names of sailors who have not reserved boat number 103.**

```
SQL> SELECT S.sname FROM Sailors S WHERE NOT EXISTS (SELECT *
FROM Reserves R WHERE R.bid = 103 AND R.sid = S.sid )  ;

SNAME
---------------
Brutus
Andy
Rusty
Horatio
Zorba
Art
Bob

7 rows selected.
```

6) **UNION**

The UNION operator is used to combine the result-set of two or
more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

Syntax:

SELECT *column_name(s)* FROM *table1*
UNION
SELECT *column_name(s)* FROM *table2*;

7. **Find the names of sailors who have reserved a red or a green boat.**

SQL> select s.sname from Sailors s,Reserves r, Boats b where s.sid = r.sid and r.bid = b.bid and b.color='red'

union

select s2.sname from Sailors s2,Reserves r2, Boats b2 where s2.sid = r2.sid and r2.bid = b2.bid and b2.color='green';

```
SNAME

---------------

Dustin

Horatio

Lubber.
```

7) INTERSECT

The **INTERSECT** operator in SQL is used to retrieve the records that are identical/common between the result sets of two SELECT (tables) statements.

Syntax:

SELECT *column_name(s)* FROM *table1*
INTERSECT
SELECT *column_name(s)* FROM *table2*;

8. **Find the names of sailors who have reserved a red or a green boat.**

SQL> select s.sname from Sailors s,Reserves r, Boats b where s.sid = r.sid and r.bid = b.bid and b.color='red'

intersect

```
select s2.sname from Sailors s2,Reserves r2, Boats b2 where s2.sid = r2.sid and
r2.bid = b2.bid and b2.color='green';
```

```
SNAME
---------------
Dustin
Horatio
Lubber
```

EXPERIMENT -3

QUERIES USING AGGREGATE FUNCTIONS (COUNT, SUM, AVG, MAX AND MIN) GROUP BY, HAVING and Creation and dropping of views.

1. Creation of emp & dept table in SQL:

```
SQL>create table dept(deptno number(2,0) primary key, dname varchar2(14)
NOT NULL, loc varchar2(13) NOT NULL);
```

Table created.

```
SQL>create table emp(empno number(4,0) primary key,ename varchar2(10)
NOT NULL, job varchar2(9) NOT NULL, mgr number(4,0),hiredate date,sal
number(7,2) NOT NULL, comm number(7,2),deptno  number(2,0),foreign key
(deptno) references dept (deptno));
```

Table created.

2. View Structure/schema of emp & dept table in SQL

```
SQL> desc emp;

Name          Null?      Type

---------------------------------------------------

EMPNO      NOT NULL NUMBER(4)

ENAME                 VARCHAR2(10)

JOB                   VARCHAR2(9)
MGR                   NUMBER(4)
HIREDATE              DATE
SAL                   NUMBER(7,2)
```

```
COMM                    NUMBER(7,2)
DEPTNO                  NUMBER(2)
```

SQL> desc dept;

```
Name              Null?  Type
--------------------------------------------------------
DEPTNO     NOT NULL  NUMBER(2)

DNAME                   VARCHAR2(14)
LOC                     VARCHAR2(13)
```

3. <u>Insert the values in emp & dept table in SQL:</u>

```
insert into dept values(10, 'ACCOUNTING', 'NEW YORK');
insert into dept values(20, 'RESEARCH', 'DALLAS');
insert into dept values(30, 'SALES', 'CHICAGO');
insert into dept values(40, 'OPERATIONS', 'BOSTON');

insert into emp values(7839, 'KING', 'PRESIDENT', null, to_date('17-11-1981','dd-mm-yyyy'),5000, null, 10);
insert into emp values(7698, 'BLAKE', 'MANAGER', 7839, to_date('1-5-1981','dd-mm-yyyy'),2850, null, 30);
insert into emp values(7782, 'CLARK', 'MANAGER', 7839, to_date('9-6-1981','dd-mm-yyyy'),2450, null, 10);
insert into emp values(7566, 'JONES', 'MANAGER', 7839,to_date('2-4-1981','dd-mm-yyyy'),2975, null, 20);
insert into emp values(7788, 'SCOTT', 'ANALYST', 7566,to_date('13-JUL-87','dd-mm-rr') - 85,3000, null, 20);
insert into emp values(7902, 'FORD', 'ANALYST', 7566, to_date('3-12-1981','dd-mm-yyyy'),3000, null, 2);
insert into emp values(7369, 'SMITH', 'CLERK', 7902, to_date('17-12-1980','dd-mm-yyyy'), 800, null, 20);
insert into emp values(7499, 'ALLEN', 'SALESMAN', 7698,to_date('20-2-1981','dd-mm-yyyy'),1600, 300, 30);
insert into emp values(7521, 'WARD', 'SALESMAN', 7698, to_date('22-2-1981','dd-mm-yyyy'),1250, 500, 30);
insert into emp values( 7654, 'MARTIN', 'SALESMAN', 7698, to_date('28-9-1981','dd-mm-yyyy'),1250, 1400, 30);
insert into emp values(7844, 'TURNER', 'SALESMAN', 7698, to_date('8-9-
1981','dd-mm-yyyy'),1500, 0, 30);
insert into emp values(7900, 'JAMES', 'CLERK', 7698,to_date('3-12-1981','dd-
mm-yyyy'),950, null, 30);
insert into emp values(7934, 'MILLER', 'CLERK', 7782,to_date('23-1-
1982','dd-mm-yyyy'),1300, null, 10);
```

4. To retrieve emp and dept values:

```
SQL> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> select * from emp;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
DEPTNO
----------
      7839 KING       PRESIDENT            17-NOV-81       5000
        10

      7698 BLAKE      MANAGER         7839 01-MAY-81       2850
        30

      7782 CLARK      MANAGER         7839 09-JUN-81       2450
        10


     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7566 JONES      MANAGER         7839 02-APR-81       2975
        20

      7788 SCOTT      ANALYST         7566 19-APR-87       3000
        20
      7369 SMITH      CLERK           7902 17-DEC-80        800
        20


     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300
        30

      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500
        30

      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400
        30


     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
---------- ---------- --------- ---------- --------- ---------- ----------
    DEPTNO
----------
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0
        30

      7900 JAMES      CLERK           7698 03-DEC-81        950
        30

      7934 MILLER     CLERK           7782 23-JAN-82       1300
        10


12 rows selected.
```

Aggregate Functions or Group Functions

An aggregate function in SQL returns one value after calculating multiple values of a column. We often use aggregate functions with the GROUP BY and HAVING clauses of the SELECT statement.

There are 5 types of SQL aggregate functions:

- Count()
- Sum()
- Avg()
- Max()
- Min()

COUNT() Function

The COUNT() function returns the number of rows in a database table.

Syntax:

COUNT(*)

or

COUNT([ALL|DISTINCT] expression)

1. COUNT: Calculate the number of employees in dept 20.

```
   SQL> SELECT COUNT (*) NO_EMP FROM EMP WHERE DEPTNO=20;
  NO_EMP
----------
       3
```

2. **COUNT THE DISTINCT EMPLOYEE SALARIES**

```
SQL> SELECT COUNT (DISTINCT SAL) NO_EMP FROM EMP;

   NO_EMP
----------
       11
```

SUM() Function

The SUM() function returns the total sum of a numeric column.

Syntax:

SUM()

or

SUM([ALL|DISTINCT] expression)

2. SUM: Calculate the total salaries for each dept

```
   SQL> SELECT DEPTNO, SUM (SAL) FROM EMP GROUP BY DEPTNO;
DEPTNO   SUM(SAL)
---------- ----------
       30       9400
       20       6775
       10       8750
```

AVG() Function

The AVG() function calculates the average of a set of values.

Syntax:

AVG()

or

AVG([ALL|DISTINCT] expression)

3. AVG: Calculate the average salaries for each dept

```
   SQL> SELECT DEPTNO, AVG (SAL) FROM EMP GROUP BY DEPTNO;
DEPTNO   AVG(SAL)
---------- ----------
      30 1566.66667
      20 2258.33333
      10 2916.66667
```

MAX() Function

The MAX() aggregate function returns the highest value (maximum) in a set of non-NULL values.

Syntax:

MAX()

or

MAX([ALL|DISTINCT] expression)

4. MAX: Calculate the maximum salary for each dept

```
   SQL> SELECT DEPTNO, MAX (SAL) FROM EMP GROUP BY DEPTNO;
DEPTNO   MAX(SAL)
---------- ----------
       30       2850
       20       3000
       10       5000
```

MIN() Function

The MIN() aggregate function returns the lowest value (minimum) in a set of non-NULL values.

Syntax:

MIN()

or

MIN([ALL|DISTINCT] expression)

5. MIN

Calculate the minimum salary for each dept

```
SQL> SELECT DEPTNO, MIN(SAL) FROM EMP GROUP BY DEPTNO
  DEPTNO   MIN(SAL)
---------- ----------
       30        950
       20        800
       10       1300
```

6. **ORDER BY Clause** :- The ORDER BY keyword is used to sort the result-set by a specified column. The ORDER BY keyword sorts the records in ascending order by default (we can even use ASC keyword). If we want to sort the records in a descending order, we can use the DESC keyword.

The general syntax is

SELECT ATT_LIST FROM TABLE_LIST ORDER BY ATT_NAMES [ASC | DESC];

```
SQL> select * from dept order by dname;


   DEPTNO DNAME          LOC
---------- -------------- -------------
       10 ACCOUNTING     NEW YORK
       40 OPERATIONS     BOSTON
       20 RESEARCH       DALLAS
       30 ALES           CHICAGO
```

7. **GROUP BY:**

The GROUP BY clause is a SQL command that is used to **group rows that have the same values**. The GROUP BY clause is used in the SELECT statement .Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.

GROUP BY Syntax

```
        SELECT statements… GROUPBY column_name1[column_name2,…];
```

```
SQL> select min(sal) from emp group by sal;

  MIN(SAL)
----------
      5000
```

```
2450

1300

2850

1250

2975

3000

 800

1600

1500

 950


11 rows selected.
```

8. HAVING

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

- The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.
- The HAVING clause must follow the GROUP BY clause in a query and must also precede the ORDER BY clause if used

HAVING Syntax

SELECT *column_name(s)* FROM *table_name* WHERE *condition* GROUP BY *column_name(s)* HAVING *condition*

```
SQL> select max(sal), deptno from emp group by deptno having
max(sal)>3000;

  MAX(SAL)     DEPTNO
---------- ----------
      5000         10
```

9. View :

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table**.**

CREATE VIEW *view_name* AS SELECT *column1*, *column2*, ...
FROM *table_name* WHERE *condition*;

```
SQL> create view department as select * from dept;

View created.
```

To retrieve the view

```
SQL> select * from department;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 PERATIONS      BOSTON
```

SQL Dropping a View

A view is deleted with the DROP VIEW statement.

SQL DROP VIEW Syntax

DROP VIEW *view_name*;

```
SQL> drop view department;


View dropped.


SQL> select * from department;
select * from department
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

EXPERIMENT -4

QUERIES USING CONVERSION FUNCTIONS (TO_CHAR, TO_NUMBER AND TO_DATE), STRING FUNCTIONS (CONCATENATION, LPAD, RPAD, LTRIM, RTRIM, LOWER, UPPER, INITCAP, LENGTH, SUBSTR AND INSTR), DATE FUNCTIONS (SYSDATE, NEXT_DAY, ADD_MONTHS, LAST_DAY, MONTHS_BETWEEN, LEAST,
GREATEST, TRUNC, ROUND, TO_CHAR)

SQL> select *from emp;

```
ENO ENAME              SALARY LOC
---------- ----------------  ---------- ----------------
101   ali              15000 vja
102   haji             20000 hyd
103   mohammad         42000 vja
104   ravi             23000 gnt
105   irfath           50000 hyd
```

a) Conversion Functions:

1. to_char: to_char is used to convert the attribute values to char.

SQL> select to_char(salary,'$99999.99') from emp;

```
TO_CHAR(SALARY)
----------
$15000.00
$20000.00
$42000.00
$23000.00
$50000.00
```

SQL> SELECT TO_CHAR (123.4567, '99999.9') FROM DUAL;

```
TO_CHAR (
--------
123.5
```

SQL> SELECT TO_CHAR(123.4567, '99999.99') FROM DUAL;

```
TO_CHAR(1

123.46
 SQL> SELECT TO_CHAR(1234.56789,'9,999.00') FROM DUAL;

TO_CHAR(1

1,234.57
SQL> SELECT TO_CHAR(SYSDATE, 'YYYY/MM/DD') FROM DUAL;

 TO_CHAR(SY

2021/07/09

SQL> SELECT TO_CHAR (SYSDATE, 'DD/MM/YYYY') FROM DUAL;

TO_CHAR(SY

09/07/2021

SQL> SELECT TO_CHAR (23, '000099') FROM DUAL;

TO_CHAR

000023

SQL> SELECT TO_CHAR (23, '0000999') FROM DUAL;

TO_CHAR(

0000023

SQL> SELECT TO_CHAR (23, '00009') FROM DUAL;

TO_CHA

00023

SQL> SELECT TO_CHAR (23, '00000') FROM DUAL;

TO_CHA

00023
```

```
SQL> SELECT TO_CHAR (234.5678, '00.00') FROM DUAL;

TO_CHA
------
 ######
SQL> SELECT TO_CHAR (234.5678, '000.000') FROM DUAL;

TO_CHAR(
--------

234.568

SQL> SELECT TO_CHAR(2345.234566, '1,23.000') FROM DUAL;
SELECT TO_CHAR(2345.234566, '1,23.000') FROM DUAL
* ERROR at line 1:
ORA-01481: invalid number format model

SQL> SELECT TO_CHAR (2345.2345, '9,000.00') FROM DUAL;

TO_CHAR(2
---------

2,345.23

SQL> SELECT TO_CHAR (2345.2345, '$9,000.00') FROM DUAL;

TO_CHAR(23
----------

$2,345.23
```

2. **to_number:** to_number is used to convert the attribute value to number.

```
SQL> SELECT TO_NUMBER('1210.73', '9999.99') FROM DUAL;
TO_NUMBER('1210.73','9999.99')
------------------------------
1210.73
```

3. **to_date:** to_date is used for convert and display the attribute values as date. SQL> select to_date('01-01-2020', 'MM-DD-YYYY') from dual;

```
TO_DATE('
---------
01-JAN-20
```

b) String functions:

1. **Concatenation:** CONCAT is used to add two attribute values such as string.

```
SQL> select concat (eno, loc) from emp;
CONCAT(ENO,LOC)
-------------------------------------------------
101    vja
102    hyd
103    vja
104    gnt
105    hyd
```

2. **lpad:** LPAD() function is used to padding the left side of a string with a specific set of characters.

```
SQL> select lpad(ename,10,'*') from emp;

LPAD(ENAME,10,'*')
---------------------------------------
*******ali
******haji
**mohammad
******ravi
****irfath
```

3. **rpad**: RPAD() function is used to padding the right side of a string with a specific set of characters.

```
SQL> select rpad(ename,10,'*') from emp;


RPAD(ENAME,10,'*')
---------------------------------------
ali*******
haji******
 mohammad**
```

```
ravi******

irfath****
```

4. **ltrim**: LTRIM() function is used to remove all specified characters from the left end side of a string

SQL> select ltrim('****hi********','*') from dual;**

```
LTRIM('***
----------

hi********
```

5. **rtrim:** RTRIM() function is used to remove all specified characters from the left end side of a string

SQL> select rtrim('****hi********','*') from dual;**

```
RTRIM('*

--------
******hi
```

6. **lower:** lower() function is used to convert the attribute value in to lower case.

SQL> select lower(ename) from emp;

```
LOWER(ENAM

----------
ali

haji
mohammad
ravi
irfath
```

7. **upper**: upper() function is used to convert the attribute values in to upper case.

SQL> select upper(ename) from emp;

```
UPPER(ENAM
----------

ALI
```

```
HAJI
MOHAMMAD
RAVI
IRFATH
```

8. **initcap**: initcap() is used to convert the attribute values first character in capital letter.

SQL> select initcap (ename) from emp;

```
INITCAP(EN
----------
Ali Haji
Mohammad
Ravi
Irfath
```

9. **length**: length() function is used to calculate the length of the given attribute.

SQL> select ename,length(ename) from emp;

```
ENAME     LENGTH(ENAME)
-------------------------

ali       3
haji      4
mohammad  8
ravi      4
irfath    6
```

10. **substr**:substr() function is used to find the substring of the given attribute value. It retuns size-1 of the given string/ attribute as a sub string.

SQL> select ename, substr(ename,4) from emp;

```
ENAME     SUBSTR(ENAME,4)
---------- ---------------
ali
haji      i
```

```
mohammad  ammad
ravi      i
irfath    ath
```

9. **instr**: instr() function return the location of starting passion of the sub string in the existing value.

SQL> select instr('welcome to CRRCOE','to') from dual;

```
INSTR('WELCOMETO CRRCOE','TO')
------------------------------

9
```

c) Date functions:

1. **Sysdate()**: sysdate() function returns the current system date.

SQL> select sysdate from dual;

```
SYSDATE
---------

 28-APR-21
```

2. **next_day()**; it reurns the date of next coming day .

SQL> select next_day(sysdate,'sunday') from dual;

```
NEXT_DAY(
---------
02-MAY-21
```

3. **add_months()**: it returns the next date after adding number of months in the orguments.

SQL> select add_months(sysdate,5) from dual;

```
 ADD_MONTH
---------
28-SEP-21
```

4. **last_day()**: The LAST_DAY() function takes a date value as argument and returns the last day of month in that date

SQL> select last_day(sysdate) from dual;

```
LAST_DAY(
---------
```

```
30-APR-21
```

```
SQL> select last_day('02-FEB-2020') from dual;
LAST_DAY(
---------
29-FEB-20
```

5. **months_between()**: it returns the numbers of months between given two dates.

```
SQL> select months_between('02-feb-2021','02-feb-2020') from dual;

MONTHS_BETWEEN('02-FEB-2021','02-FEB-2020')
-------------------------------------------
12

SQL> select months_between(sysdate,'02-feb-2020') from dual;

MONTHS_BETWEEN(SYSDATE,'02-FEB-2020')
-------------------------------------
14.8600769
```

6. **least()**: it retuns least value from the given argument or attributes.

```
SQL> select least(300,450,100,440) from dual;
LEAST(300,450,100,440)
----------------------
100
```

7. **greatest()**: it returns maximum values from the given arguments or attributes in the relation.

```
SQL> select greatest(300,450,100,440) from dual;
GREATEST(300,450,100,440)
-------------------------
450
```

8. **trunc()**: The TRUNC() function returns a DATE value truncated to a specified unit.

```
SQL> select trunc(sysdate,'mm') from dual;

TRUNC(SYS
---------
01-APR-21
SQL> select trunc(sysdate,'yyyy') from dual;
```

```
TRUNC(SYS
---------
01-JAN-21
```

9. **round()**: Round function round a number to a specified length or precision.

SQL> select round(12.49,0) from dual;

```
ROUND(12.49,0)
--------------
12
```

SQL> select round(12.51,0) from dual;

```
 ROUND(12.51,0)
 --------------
13
```

10. **to_char()**: it convert the given date type attribute values to text and return the date in the specific format.

SQL> select to_char(sysdate,'yyyy-mm-dd') from dual;

```
 TO_CHAR(SY
----------
2021-04-28
```

PL/SQL

- PL/SQL stands for Procedural Language extensions to the Structured Query Language (SQL). SQL is a powerful language for both querying and updating data in relational databases.
- The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as procedural extension language for SQL and the Oracle relational database.
- Basic Syntax of PL/SQL, a **block-structured** language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts –

S.No	Sections & Description
1	**Declarations** This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.
2	**Executable Commands** This section is enclosed between the keywords **BEGIN** and **END** and is mandatory. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a **NULL command** to indicate that nothing should be executed.
3	**Exception Handling** This section starts with the keyword **EXCEPTION**. This optional section contains **exception(s)** that handle errors in the program.

Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using **BEGIN** and **END**. Following is the basic structure of a PL/SQL block −

Basic structure of a PL/SQL block

```
DECLARE
<declarations section>
BEGIN
<executable command(s)>
EXCEPTION
<exception handling>
END;
```

EXPERIMENT: 5

i. Create a simple PL/SQL program which includes declaration section, executable section and exception –Handling section (Ex. Student marks can be selected from the table and printed for those who secured first class and an exception can be raised if no records were found).

ii. Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block..

i). We have to create the student table and insert the records in to the table as follows:

```
SQL> create table student(sid number(10),sname varchar2(20),rank
varchar(10));

     Table created.

SQL> insert into student values(501,'Ravi','second');

     1 row created.

SQL> insert into student values(502,'Raju','third');

     1 row created.

SQL> insert into student values(503,'Ramu','');

     1 row created.

SQL> select *from student;

 SID SNAME                RANK
---------- -------------------- ----------
       501 Ravi                 second
       502 Raju                 third
       503 Ramu
```

i. Create a simple PL/SQL program which includes declaration section, executable section and exception –Handling section (Ex. Student marks can be selected from the table and printed for those who secured first class and an exception can be raised if no records were found).

● Type your PL/SQL code in a text editor like Notepad, Notepad++, or EditPlus.
● Save the file with the **.sql** extension in the home directory.
● Launch the **SQL*Plus command prompt** from the directory where you created your PL/SQL file.
● Type **@file_name** at the SQL*Plus command prompt to execute your program.

<u>PL/SQL CODE:</u>

```
set serveroutput on;

declare

temp1 number(10);

temp2 varchar2(10);

begin

select sid,sname into temp1,temp2 from student where rank='first';
dbms_output.put_line('Student No:'|| temp1 ||' Name:'||temp2||' got first rank');

exception

When no_data_found then dbms_output.put_line('***********************');
dbms_output.put_line('# Error: there is no student got first rank');

end;

/
```

Output:

```
SQL> @5a;

**************************

# Error: there is no student got first rank
```

```
PL/SQL procedure successfully completed.


SQL> update student set rank='first' where sid=503;

1 row updated.

SQL> select *from student;

 SID SNAME              RANK
---------- -------------------- ----------
       501 Ravi               second
       502 Raju               third
       503 Ramu                 first


SQL> @5a
Student No:503  Name:Ramu  got first rank

PL/SQL procedure successfully completed.
```

ii) Insert data into student table and use COMMIT, ROLLBACK and SAVEPOINT in PL/SQL block..

```
        SQL> select *from student;
 SID SNAME              RANK
---------- -------------------- ----------
      501 Ravi              second
      502 Raju              third
      503 Ramu                first
```

PL/SQL CODE:

```
set serveroutput on;
DECLARE
sno student.sid%type;
name student.sname%type;
srank student.rank%type;

BEGIN
sno := &sno;
name := '&name';
srank := '&srank';
INSERT into student values(sno,name,srank);
dbms_output.put_line('One record inserted');
COMMIT;
-- adding savepoint
SAVEPOINT s1;
-- second time asking user for input
sno := &sno;
name := '&name';
srank := '&srank';
```

```
INSERT into student values(sno,name,srank);

dbms_output.put_line('One record inserted');

ROLLBACK TO SAVEPOINT s1;

END;

/
```

```
 SQL> @5b
 Enter value for sno: 504
 old   7:sno := &sno;
 new   7:sno := 504;

 Enter value for name: ali
old   8:         name := '&name';

new   8:name := 'ali';

Enter value for srank: first

old   9:         srank := '&srank';

new   9:srank := 'first';

 Enter value for sno: 505

 old  16:sno := &sno;

 new  16:                sno := 505;
 Enter value for name: haji

 old  17:name := '&name';

 new  17:name := 'haji';

  Enter value for srank: third

 old  18:srank := '&srank';

 new  18:srank := 'third';

One record inserted
One record inserted

PL/SQL procedure successfully completed.

SQL> select *from student;

 SID SNAME               RANK
```

```
---------- -------------------- ----------
       501 Ravi                 second
       502 Raju                 third
       503 Ramu                   first
       504 suresh                first
```

EXPERIMENT: 6

Develop a program that includes the features NESTED IF, CASE and CASE expression. The program can be extended using the NULLIF and COALESCE functions.

A. NESTED IF:

A nested if-then is an if statement that is the target of another if statement. Nested if -then statements mean an if statement inside another if statement

Syntax:-

```
if (condition1) then
-- Executes when condition1 is true if (condition2) then
-- Executes when condition2 is true end if;
end if;
```

PL/SQL CODE: PL/SQL Program to find biggest of three number using nested if.

```
SQL>

Set serveroutput on;
declare
        a number:=10;
        b number:=12;
        c number:=5;
begin
        dbms_output.put_line('a='||a||' b='||b||' c='||c);
        if a>b AND a>c then
                dbms_output.put_line('a is greatest');
        else
                if b>a AND b>c then
                        dbms_output.put_line('b is greatest');
                else
                        dbms_output.put_line('c is greatest');
                end if;
        end if;
end;
/
```

SQL> @E:\GSK\largest.sql

a=10 b=12 c=5

```
b is greatest
PL/SQL procedure successfully completed.
```

B. CASE and CASE Expression : CASE statement selects one sequence of statements to execute. However, to select the sequence, the **CASE** statement uses a selector rather than multiple Boolean expressions. A selector is an expression, the value of which is used to select one of several alternatives.

Syntax

```
CASE selector
WHEN 'value1' THEN S1;
WHEN 'value2' THEN S2;
WHEN 'value3' THEN S3;
...
ELSE Sn; -- default case
END CASE;
```

```
SQL> create table emp1(eno number(5), ename varchar2(10), loc varchar(10),
salary number(10,2));
 Table created.

SQL> insert into emp values(101,'ali','vja',15000);
1 row created.
SQL> insert into emp1 values(102,'ravi','hyd',25000);
1 row created.
SQL> insert into emp1 values(103,'raju','gnt',35000);
1 row created.
SQL> insert into emp1 values(104,'rakesh','vja',45000);
1 row created.

SQL> select *from emp1;

ENO    ENAME        LOC    SALARY
------ ----------   ------ ------------
101    ali          vja    15000
102    ravi         hyd    25000
103    raju         gnt    35000
104    rakesh              vja     45000
```

Example of CASE Expression:

```
SQL> select loc, case(loc) when 'vja' then salary+2000 when 'hyd' then
salary+1000 else salary
end "rev_salary" from emp;

LOC rev_salary
------  ------------
vja     17000
hyd     26000
gnt     35000
vja     47000
```

PL/SQL CODE: PL/SQL CODE to demonstrate CASE

```
SQL>
set serveroutput on;
declare
        grade char(1);
begin
grade:='&grade';
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;
/
```

```
SQL>  @E:\GSK\grade.sql
Enter value for grade: c old 4: grade:='&grade';
new 4: grade:='c'; good
```

```
PL/SQL procedure successfully completed.

SQL> @6b
Enter value for grade: g old 4: grade:='&grade';
new 4: grade:='g';
No such grade
PL/SQL procedure successfully completed.
```

C. NULLIF: Takes two arguments. If the two arguments are equal, then NULL is returned. otherwise the first argument is returned.

Syntax: select column_name, NULLIF(argument1,arguement2) from table_name;

Example:

```
SQL> select ename, nullif('ali','ali1') from emp1;
ENAME NUL
--------          ------
ali     ali
ravi    ali
raju    ali
rakesh ali
SQL> select ename, nullif('ali','ali') from emp1;
ENAME NUL
---------         -------
ali
ravi
raju
rakesh
```

D. COALESCE: COALESCE () function accepts a list of arguments and returns the first one that evaluates to a non-null value.

Syntax: coalesce("expression1","expression2",...);

Example:

```
SQL> select coalesce(NULL,'NAME','CSM') from dual;
COALE
-----------
NAME
```