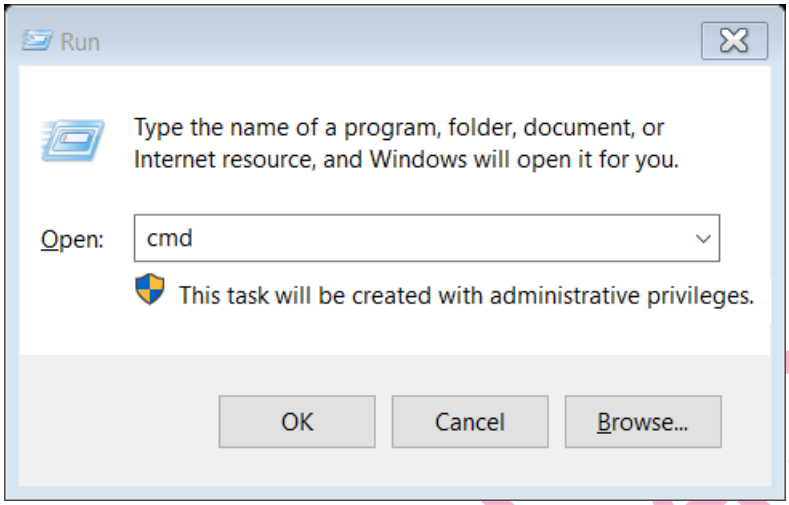
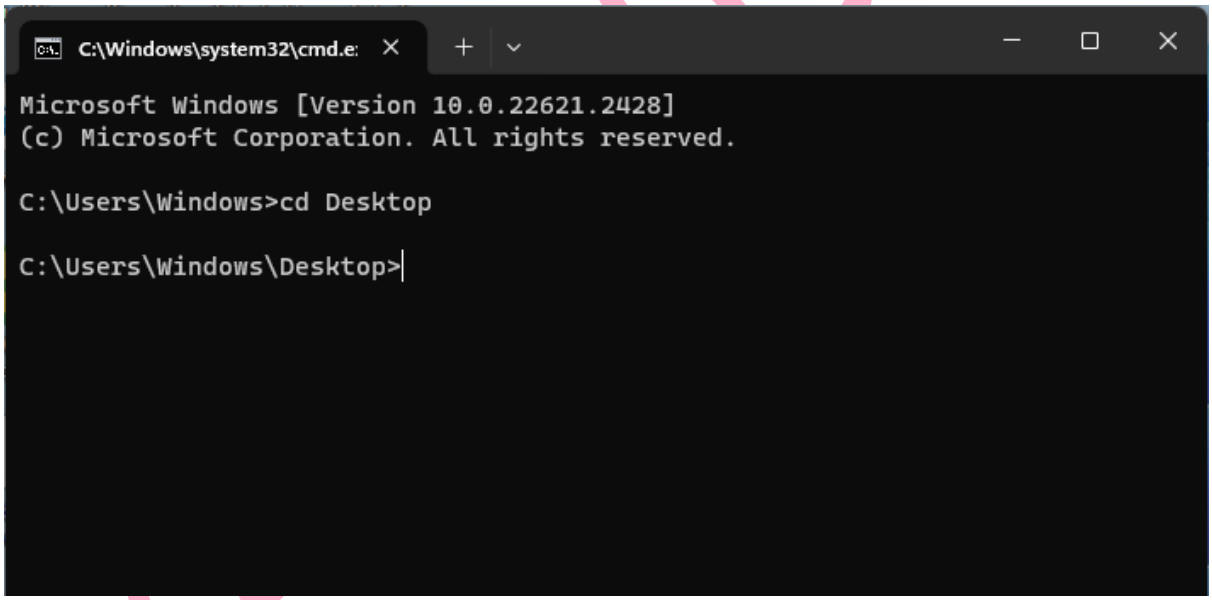
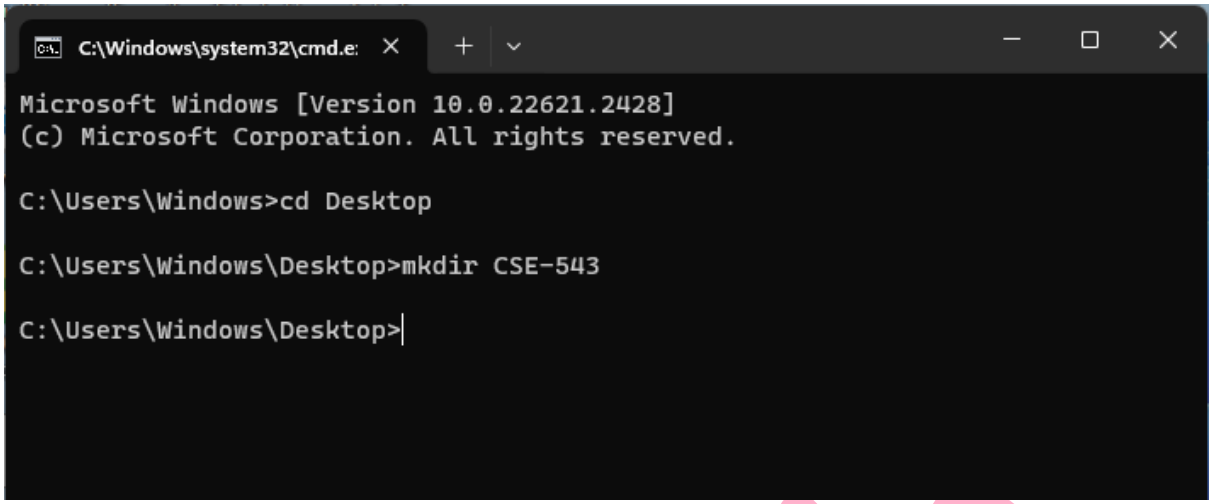
1. Open the command prompt Press WIN+R type cmd.



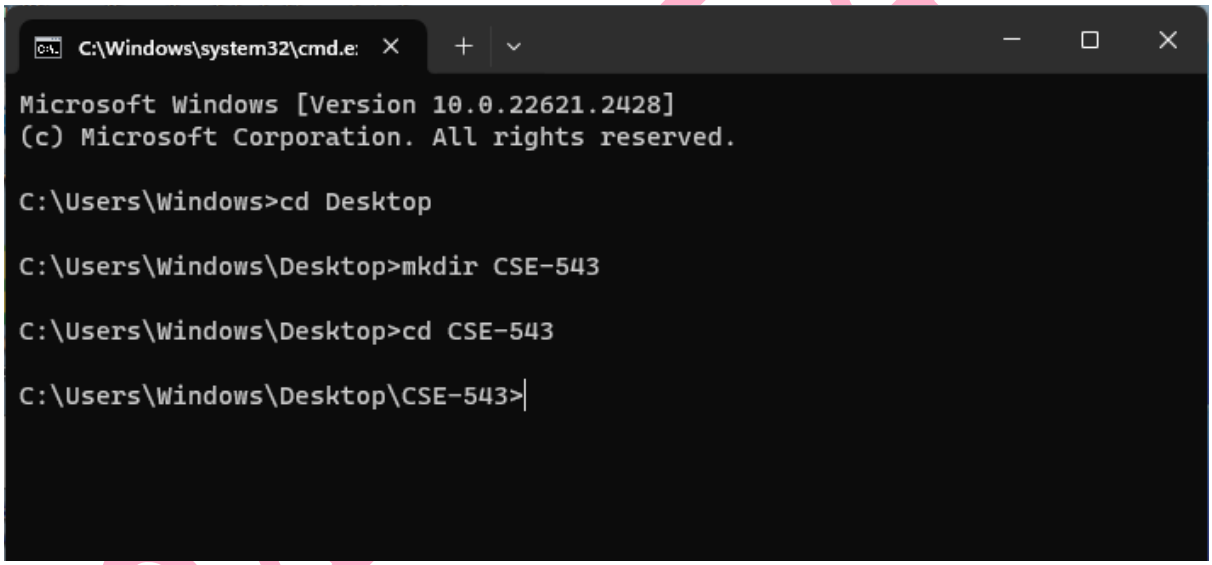
2. Once cmd prompt open go to DESKTOP using cd Desktop.



3. Now create a Directory using mkdir or md command using your branch abbreviation and last 3 digits hall ticket number like md cse-543.



4. Now, move into the directory by using cd command show below.



Experiment -1

31-AUG-2023

SQL queries to CREATE TABLES for various databases using DDL commands (CREATE, ALTER, DROP, TRUNCATE)

5. Create a table parts with attributes part_id, part_name, unique_code, manufactured_date and cost.

```
SQL> CREATE TABLE parts(
  2  part_id NUMBER PRIMARY KEY,
  3  part_name VARCHAR2(50) NOT NULL,
  4  unique_code VARCHAR2(20) NOT NULL,
  5  manufactured_date DATE NOT NULL,
  6  cost NUMBER(10,2) NOT NULL
  7  );

Table created.

SQL>
```

6. Check the created table attributes using "DESC parts" command.

```
SQL> DESC parts;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PART_ID                                   NOT NULL NUMBER
 PART_NAME                                 NOT NULL VARCHAR2(50)
 UNIQUE_CODE                               NOT NULL VARCHAR2(20)
 MANUFACTURED_DATE                         NOT NULL DATE
 COST                                      NOT NULL NUMBER(10,2)

SQL> |
```

7. Alter the table parts using alter command.

```
SQL> ALTER TABLE parts
  2  ADD quantity NUMBER NOT NULL;

Table altered.
```

8. Check table whether quantity is added to table or not .

```
SQL> DESC parts;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PART_ID                                   NOT NULL NUMBER
 PART_NAME                                 NOT NULL VARCHAR2(50)
 UNIQUE_CODE                               NOT NULL VARCHAR2(20)
 MANUFACTURED_DATE                         NOT NULL DATE
 COST                                      NOT NULL NUMBER(10,2)
 QUANTITY                                  NOT NULL NUMBER

SQL> |
```

9. Now alter the manufactured_date using alter command.

```
SQL> ALTER TABLE parts
  2  MODIFY manufactured_date DATE;

Table altered.

SQL> |
```

10. Check whether manufactured_date is altered or not.

```
SQL> DESC parts;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PART_ID                                   NOT NULL NUMBER
 PART_NAME                                 NOT NULL VARCHAR2(50)
 UNIQUE_CODE                               NOT NULL VARCHAR2(20)
 MANUFACTURED_DATE                         NOT NULL DATE
 COST                                      NOT NULL NUMBER(10,2)
 QUANTITY                                  NOT NULL NUMBER

SQL> |
```

11. Remove manufactured_date column from table using drop command.

```
SQL> ALTER TABLE parts
  2  DROP COLUMN manufactured_date;

Table altered.

SQL> |
```

12. Check whether column is dropped or not.

```
SQL> DESC parts;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PART_ID                                   NOT NULL NUMBER
 PART_NAME                                 NOT NULL VARCHAR2(50)
 UNIQUE_CODE                               NOT NULL VARCHAR2(20)
 COST                                      NOT NULL NUMBER(10,2)
 QUANTITY                                  NOT NULL NUMBER

SQL> |
```

13. Create a table boat with attributes boat_id, boat_name.

```
SQL> CREATE TABLE boats(
  2  boat_id NUMBER PRIMARY KEY,
  3  boat_name VARCHAR2(50) NOT NULL
  4  );

Table created.

SQL> |
```

14. Drop table boat.

```
SQL> DROP TABLE boats;

Table dropped.
```

15. Insert values into Part table.

```
SQL> INSERT INTO parts VALUES (101,'XAT','XHJ45689023K',999.89,98);

1 row created.

SQL> |
```

16. Display table using “SELECT * FROM parts;” command.

```
SQL> SELECT * FROM parts;

   PART_ID PART_NAME
---------- --------------------------------------------------
UNIQUE_CODE                COST   QUANTITY
-------------------- ---------- ----------
       101 XAT
XHJ45689023K               999.89         98


SQL> |
```

17. Delete the table parts.

```
SQL> TRUNCATE TABLE parts;

Table truncated.
```

18. Check whether table is deleted or not.

```
SQL> DESC boats;
ERROR:
ORA-04043: object boats does not exist


SQL> |
```

19.Summary of the Lab Report

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	14
2.	Number of tables creation specified in observation	2
3.	Number of tables you created in the lab	2
4.	Number of Select Statements specified in the observation	5
5.	Number of Select statements you practised in lab	5
6	Number of Insert Statements specified in observation	3
7	Number of Insert Statements you practiced in Lab	3
8.	Number of Alter Statements specified in observation	5
9	Number of Alter Statements practiced in lab	5
10	Total number of Statements specified in lab	20
11	Total number of statements practiced in lab	20

Experiment-2

05-OCT-2023

SQL queries to MANIPULATE TABLES for various databases using DML commands (INSERT, SELECT, UPDATE, DELETE)

1. Create a table person with attributes person_id, first_name, last_name, phone_no.

```
SQL>
SQL> CREATE TABLE person(
  2  person_id NUMBER PRIMARY KEY,
  3  First_name VARCHAR2(50) NOT NULL,
  4  Last_name VARCHAR2(50) NOT NULL,
  5  phone_no NUMBER(10) NOT NULL
  6  );

Table created.

SQL>
```

2. Check the attributes of person table.

```
SQL> DESC person;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PERSON_ID                                 NOT NULL NUMBER
 FIRST_NAME                                NOT NULL VARCHAR2(50)
 LAST_NAME                                 NOT NULL VARCHAR2(50)
 PHONE_NO                                  NOT NULL NUMBER(10)

SQL> |
```

3. Insert rows into person table.

```
SQL> INSERT INTO person(person_id,first_name,last_name,phone_no) VALUES(501,'Ravi','Krishna',9128730465);

1 row created.

SQL> INSERT INTO person(person_id,first_name,last_name,phone_no) VALUES(502,'Chandra','Sekhar',8237195064);

1 row created.

SQL> INSERT INTO person(person_id,first_name,last_name,phone_no) VALUES(503,'Ramesh','Kumar',7193506428);

1 row created.

SQL> |
```

4. Display person table using 'SELECT * FROM person' command.

```
SQL> SELECT * FROM person;

 PERSON_ID FIRST_NAME
---------- ------------------------------------------------
LAST_NAME                                           PHONE_NO
-------------------------------------------------- ----------
       501 Ravi
Krishna                                            9128730465

       502 Chandra
Sekhar                                             8237195064

       503 Ramesh
Kumar                                              7193506428
```

5. Create table discounts with attributes offer, start_date, end_date.

```
SQL> CREATE TABLE discounts(
  2  discount_id NUMBER PRIMARY KEY,
  3  discount_name VARCHAR2(50) NOT NULL,
  4  offer NUMBER(4,2),
  5  start_date DATE NOT NULL,
  6  end_date DATE NOT NULL,
  7  check(end_date>start_date)
  8  );

Table created.

SQL>
```

6. Check attributes of discounts table.

```
SQL> DESC discounts;
 Name
  Null?    Type
 ----------------------------------------------------------------------------------------------------------------------------
 ----------------------------
 DISCOUNT_ID
  NOT NULL NUMBER
 DISCOUNT_NAME
  NOT NULL VARCHAR2(50)
 OFFER
   NUMBER(4,2)
 START_DATE
  NOT NULL DATE
 END_DATE
  NOT NULL DATE

SQL>
```

7. Insert rows into discounts table.

```
SQL> INSERT INTO discounts VALUES (1,'Christmas sales',15.5,DATE'2023-11-10',DATE'2023-11-15');

1 row created.

SQL> INSERT INTO discounts VALUES (2,'New Year sales',15.5,DATE'2023-11-10',DATE'2023-11-15');

1 row created.

SQL>
```

8. Display table discounts.

```
SQL> SELECT * FROM discounts;

DISCOUNT_ID DISCOUNT_NAME                                          OFFER
----------- ------------------------------------------------ ----------
START_DAT END_DATE
--------- ---------
          1 Christmas sales                                         15.5
10-NOV-23 15-NOV-23

          2 New Year sales                                          15.5
10-NOV-23 15-NOV-23


SQL> |
```

9. Create table original bill with attributes product_no, product_name, quantity, cost.

```
SQL> CREATE TABLE original_bill(
  2  produCt_no NUMBER PRIMARY KEY,
  3  product_name VARCHAR2(110) NOT NULL,
  4  quantity NUMBER NOT NULL,
  5  cost NUMBER(10,2) NOT NULL
  6  );

Table created.

SQL> |
```

10. Check attributes of table original bill using 'DESC original bill' command.

```
SQL>
SQL> DESC original_bill;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODUCT_NO                                NOT NULL NUMBER
 PRODUCT_NAME                              NOT NULL VARCHAR2(110)
 QUANTITY                                  NOT NULL NUMBER
 COST                                      NOT NULL NUMBER(10,2)

SQL> |
```

11. Insert values into original bill table.

```
SQL> INSERT ALL
  2  INTO original_bill(product_no,product_name,quantity,cost) VALUES(1,'ABC',2,29.9)
  3  INTO original_bill(product_no,product_name,quantity,cost) VALUES(2,'AB34',5,89)
  4  INTO original_bill(product_no,product_name,quantity,cost) VALUES(3,'JK98',1,75)
  5  INTO original_bill(product_no,product_name,quantity,cost) VALUES(4,'KL77',2,99)
  6  INTO original_bill(product_no,product_name,quantity,cost) VALUES(5,'NM87',1,50)
  7  SELECT * FROM dual;

5 rows created.

SQL> |
```

12. Display attributes of original bill.

```
C:\Windows\system32\cmd.e:

5 rows created.

SQL> SELECT * FROM original_bill;

PRODUCT_NO
----------
PRODUCT_NAME
--------------------------------------------------------------------------------
  QUANTITY       COST
---------- ----------
         1
ABC
         2       29.9

         2
AB34
         5         89

PRODUCT_NO
----------
PRODUCT_NAME
--------------------------------------------------------------------------------
  QUANTITY       COST
---------- ----------

         3
JK98
         1         75

         4
KL77

PRODUCT_NO
----------
PRODUCT_NAME
--------------------------------------------------------------------------------
  QUANTITY       COST
---------- ----------
         2         99

         5
NM87
         1         50


SQL> |
```

13. Create table copy bill with attributes product no, product name, cost.

```
SQL> CREATE TABLE original_bill(
  2  produCt_no NUMBER PRIMARY KEY,
  3  product_name VARCHAR2(110) NOT NULL,
  4  quantity NUMBER NOT NULL,
  5  cost NUMBER(10,2) NOT NULL
  6  );

Table created.

SQL> |
```

14. Insert values of original table into copy bill table.

```
SQL> INSERT INTO copy_bill
  2  SELECT * FROM original_bill;

5 rows created.

SQL> |
```

15. Display copyl bill table.

```
C:\Windows\system32\cmd.e:

SQL> SELECT * FROM copy_bill;

PRODUCT_NO PRODUCT_NAME                                        QUANTITY
---------- ------------------------------------------------------------ ----------
      COST
----------
         1 ABC
      2
        70

         2 AB34
      5
        89

         3 JK98
      1
        75

PRODUCT_NO PRODUCT_NAME                                        QUANTITY
---------- ------------------------------------------------------------ ----------
      COST
----------
         4 KL77
      2
        99

         5 NM87
      1
        50

SQL>
```

16. Update row of copy bill table.

```
SQL> UPDATE copy_bill
  2  SET cost=70
  3  WHERE product_no=1;

1 row updated.

SQL>
```

17. Check the updated row.

```
SQL> SELECT * FROM copy_bill WHERE product_no=1;

PRODUCT_NO PRODUCT_NAME                                        QUANTITY
---------- -------------------------------------------------- ----------
      COST
----------
         1 ABC
      2
        70

SQL>
```

18. Update all the rows of copy bill table.

```
SQL> SELECT * FROM copy_bill;

PRODUCT_NO PRODUCT_NAME                                        QUANTITY
---------- -------------------------------------------------- ----------
      COST
----------
         1 ABC                                                         2
      29.9

         2 AB34                                                        5
        89

         3 JK98                                                        1
        75


PRODUCT_NO PRODUCT_NAME                                        QUANTITY
---------- -------------------------------------------------- ----------
      COST
----------
         4 KL77                                                        2
        99

         5 NM87                                                        1
        50


SQL>
```

19. Delete a row from the table copy bill.

```
SQL> DELETE FROM copy_bill WHERE cost>120;

2 rows deleted.

SQL>
```

20. Check whether row is deleted or not.

```
SQL> SELECT * FROM copy_bill;

PRODUCT_NO PRODUCT_NAME                                        QUANTITY
---------- ------------------------------------------------ ----------
      COST
----------
         1 ABC                                                       2
       105

         3 JK98                                                      1
     112.5

         5 NM87                                                      1
        75


SQL> |
```

21.Summary of the Lab Report

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	20
2.	Number of tables creation specified in observation	5
3.	Number of tables you created in the lab	5
4.	Number of Select Statements specified in the observation	10
5.	Number of Select statements you practised in lab	10
6.	Number of Insert Statements specified in observation	10
7.	Number of Insert Statements you practiced in Lab	10
8.	Total number of Statements specified in lab	20
9.	Total number of statements practiced in lab	20

Experiment-3

19-OCT-23

SQL queries to view various databases

(CREATE VIEW, INSERT VIEW, UPDATE VIEW, DROP VIEW)

1. Now create a table student with attributes name, rollno, course.

```
SQL> CREATE TABLE student (
  2  NAME VARCHAR2(20),
  3  ROLLNO NUMBER,
  4  COURSE VARCHAR2(20)
  5  );

Table created.

SQL>
```

2. Insert values into table student.

```
SQL> INSERT INTO student VALUES('Anu','501','cse');

1 row created.

SQL> INSERT INTO student VALUES('Mani','502','csm');

1 row created.

SQL> INSERT INTO student VALUES('Moni','503','csd');

1 row created.

SQL>
```

3. Using the command "SELECT * FROM student;" display table student.

```
SQL> SELECT * FROM student;

NAME                    ROLLNO COURSE
-------------------- ---------- --------------------
Anu                         501 cse
Mani                        502 csm
Moni                        503 csd

SQL>
```

4. CREATE VIEW: Create a view teacher with attributes name and rollno.

```
SQL> CREATE VIEW teacher AS SELECT NAME,ROLLNO FROM student;

View created.

SQL> |
```

5. Insert values into teacher view.

```
SQL> INSERT INTO teacher(NAME,ROLLNO) VALUES('Vani','523');

1 row created.

SQL> INSERT INTO teacher(NAME,ROLLNO) VALUES('Hema','537');

1 row created.

SQL> |
```

6. Display view teacher using SELECT * FROM teacher command.

```
SQL> SELECT * FROM teacher;

NAME                     ROLLNO
-------------------- ----------
Anu                         501
Mani                        502
Moni                        503
Vani                        523
Hema                        537

SQL> |
```

7. INSERT VIEW: Create view details with attributes name, rollno, course of specific attribute provided.

```
SQL> CREATE VIEW DETAILS AS SELECT NAME,ROLLNO,COURSE FROM student WHERE rollno=523;

View created.

SQL> |
```

8. Display the specific attribute provided.

```
SQL> SELECT * FROM details WHERE ROLLNO=523;

NAME                     ROLLNO COURSE
-------------------- ---------- --------------------
Vani                        523

SQL> |
```

9. UPDATE VIEW: Update the existing name with the new name using UPDATE command.

```
SQL> UPDATE teacher SET NAME='Mounika' WHERE ROLLNO=503;

1 row updated.

SQL>
```

10. Display the updated view.

```
SQL> SELECT * FROM teacher;

NAME                    ROLLNO
------------------- ----------
Anu                        501
Mani                       502
Mounika                    503
Vani                       523
Hema                       537

SQL>
```

11. DROP VIEW: Drop a view using the command DROP.

```
SQL> DROP VIEW details;

View dropped.

SQL>
```

12. Check whether view is droped or not.

```
SQL> SELECT * FROM details;
SELECT * FROM details
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL>
```

13. Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	12
2.	Number of tables creation specified in observation	1
3.	Number of tables you created in the lab	1

4.	Number of Select Statements specified in the observation	5
5.	Number of Select statements you practised in lab	5
6	Number of Insert Statements specified in observation	5
7	Number of Insert Statements you practiced in Lab	5
8.	Number of views created in observation	2
9	Number of views created in lab	2
10	Total number of Statements specified in lab	20
11	Total number of statements practiced in lab	20
12	Number of any addition statements practiced by you.	3

Experiment-4

9-NOV-2023

SQL queries to perform RELATIONAL OPERATIONS

(UNION, UNION ALL, INTERSECT, MINUS, CROSS JOIN)

1. Now create a table instructor with attributes id, name, dept_name, salary.

```
SQL> CREATE TABLE instructor (
  2  id NUMBER(10) PRIMARY KEY,
  3  name VARCHAR2(25) NOT NULL,
  4  dept_name VARCHAR2(25) NOT NULL,
  5  salary NUMBER(10,2) NOT NULL
  6  );

Table created.

SQL>
```

2. Insert values into table instructor.

```
SQL> INSERT INTO instructor VALUES('543','Madhuri','cse','73000');

1 row created.

SQL> INSERT INTO instructor VALUES('523','Mounika','csm','45000');

1 row created.

SQL> INSERT INTO instructor VALUES('565','Harika','csd','32000');

1 row created.

SQL>
```

3. Using the command "SELECT * FROM instructor;" display table instructor.

```
SQL> SELECT * FROM instructor;

        ID NAME                      DEPT_NAME                    SALARY
---------- ------------------------- ------------------------- ----------
       543 Madhuri                   cse                           73000
       523 Mounika                   csm                           45000
       565 Harika                    csd                           32000

SQL>
```

4. Create a table department with attributes dept_id, dept_name, building, budget.

```
SQL> CREATE TABLE department (
  2  dept_id VARCHAR2(10) PRIMARY KEY,
  3  dept_name VARCHAR2(10) NOT NULL,
  4  building VARCHAR2(20),
  5  budget NUMBER(10,2) NOT NULL
  6  );

Table created.

SQL>
```

5. Insert values into department table.

```
Table created.

SQL> INSERT INTO department VALUES('101','cse','A-Block','78000');

1 row created.

SQL> INSERT INTO department VALUES('102','civil','B-Block','85000');

1 row created.

SQL> INSERT INTO department VALUES('103','mech','C-Block','93000');

1 row created.

SQL>
```

6. UNION: The attributes dept_name from instructor and department are joined using command UNION.

```
SQL> SELECT dept_name FROM instructor
  2  UNION
  3  SELECT dept_name FROM department;

DEPT_NAME
------------------------
cse
csm
csd
civil
mech

SQL>
```

7. UNION ALL: The attributes dept_name from instructor and department are joined along with the duplicates using command UNION ALL.

```
SQL> SELECT dept_name FROM instructor
  2  UNION ALL
  3  SELECT dept_name FROM department;

DEPT_NAME
------------------------
cse
csm
csd
cse
civil
mech

6 rows selected.

SQL>
```

8. INTERSECT: Displays similar values in two or more attributes from department and instructor using command INTERSECT.

```
SQL> SELECT dept_name FROM department
  2  INTERSECT
  3  SELECT dept_name FROM instructor;

DEPT_NAME
------------------------
cse

SQL>
```

9. MINUS: It eliminates the same values of second column from the first column and represents the remaining values using commnd MINUS.

```
SQL> SELECT dept_name FROM department
  2  MINUS
  3  SELECT dept_name FROM instructor;

DEPT_NAME
------------------------
civil
mech

SQL>
```

10. CROSS JOIN: It cross products the all the attributes using command CROSS JOIN.

```
SQL> SELECT i.name,d.dept_name,d.budget
  2  FROM instructor i,department d;

NAME                      DEPT_NAME      BUDGET
------------------------- ---------- ----------
Madhuri                   cse             78000
Mounika                   cse             78000
Harika                    cse             78000
Madhuri                   civil           85000
Mounika                   civil           85000
Harika                    civil           85000
Madhuri                   mech            93000
Mounika                   mech            93000
Harika                    mech            93000

9 rows selected.

SQL>
```

11.Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	9
2.	Number of tables creation specified in observation	2
3.	Number of tables you created in the lab	2
4.	Number of Select Statements specified in the observation	2
5.	Number of Select statements you practised in lab	2

6	Number of Insert Statements specified in observation	6
7	Number of Insert Statements you practiced in Lab	6
8	Total number of Statements specified in lab	20
9	Total number of statements practiced in lab	20
10	Number of any addition statements practiced by you.	3

Experiment-5

16-NOV-2023

SQL queries to perform SPECIAL OPERATIONS

(IS NULL, BETWEEN, LIKE, IN, EXISTS)

1. Now create a table instructor with attributes id, name,dept_name, salary.

```
SQL> CREATE TABLE instructor (
  2  ID VARCHAR2(5),
  3  NAME VARCHAR2(20) NOT NULL,
  4  DEPT_NAME VARCHAR2(20),
  5  SALARY NUMERIC(8,2)
  6  );

Table created.

SQL>
```

2. Insert values into table instructor.

```
SQL> INSERT INTO instructor VALUES('501','Raju','csm','29001');

1 row created.

SQL> INSERT INTO instructor VALUES('502','Ramu','csd','30000');

1 row created.

SQL> INSERT INTO instructor VALUES('503','Ravi','cse','30000');

1 row created.

SQL> INSERT INTO instructor VALUES('504','Suresh','civil','56000');

1 row created.

SQL> INSERT INTO instructor VALUES('505','Ramesh','mech','54000');

1 row created.

SQL> INSERT INTO instructor VALUES('506','Anu','','');

1 row created.

SQL> INSERT INTO instructor VALUES('507','Mani','','');

1 row created.

SQL>
```

3. Using the command "SELECT * FROM instructor;" display table instructor.

```
SQL> SELECT * FROM instructor;

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
501   Raju                 csm                       29001
502   Ramu                 csd                       30000
503   Ravi                 cse                       30000
504   Suresh               civil                     56000
505   Ramesh               mech                      54000
506   Anu
507   Mani

7 rows selected.

SQL> |
```

4. IS NULL: It is used to check null values and display null attributes.
   It displays attributes that have null values.

```
SQL> SELECT * FROM instructor WHERE salary IS NULL;

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
506   Anu
507   Mani

SQL> |
```

5. This command displays the salary that are not equal to 30000.

```
SQL> SELECT * FROM instructor WHERE salary <>30000;

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
501   Raju                 csm                       29001
504   Suresh               civil                     56000
505   Ramesh               mech                      54000

SQL> |
```

6. IS NOT NULL: It displays attributes that don't have null values.

```
SQL> SELECT * FROM instructor WHERE salary IS NOT NULL;

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
501   Raju                 csm                       29001
502   Ramu                 csd                       30000
503   Ravi                 cse                       30000
504   Suresh               civil                     56000
505   Ramesh               mech                      54000

SQL> |
```

7. BETWEEN: This is used to check range of values.
   By the following command it displays all the attributes between 20000 and 30000.

```
SQL> SELECT * FROM instructor WHERE salary BETWEEN 20000 AND 30000;

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
501   Raju                 csm                       29001
502   Ramu                 csd                       30000
503   Ravi                 cse                       30000

SQL> |
```

8. The following command displays salary that are not between 20000 and 30000.

```
SQL> SELECT * FROM instructor WHERE salary NOT BETWEEN 20000 AND 30000;

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
504   Suresh               civil                     56000
505   Ramesh               mech                      54000

SQL> |
```

9. IN: This is used to check a member is in a set or not.
   It displays if the id's are present in the table.

```
SQL> SELECT * FROM instructor WHERE ID IN ('504','501','502');

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
501   Raju                 csm                       29001
502   Ramu                 csd                       30000
504   Suresh               civil                     56000

SQL> |
```

10. The following command displays all the attributes with id's except the given id's.

```
SQL> SELECT * FROM instructor WHERE ID NOT IN ('504','501','502');

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
503   Ravi                 cse                       30000
505   Ramesh               mech                      54000
506   Anu
507   Mani

SQL> |
```

11. EXITS: This is used to check whether given set is empty or not.
    It displays null attributes that are null according to the given condition.

```
SQL> SELECT * FROM instructor WHERE EXISTS
  2  (SELECT * FROM instructor WHERE dept_name IS NULL);

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
501   Raju                 csm                       29001
502   Ramu                 csd                       30000
503   Ravi                 cse                       30000
504   Suresh               civil                     56000
505   Ramesh               mech                      54000
506   Anu
507   Mani

7 rows selected.

SQL> |
```

12. LIKE: This is used to check given string is present or not.
    It displays all the attributes that start with character 'c'.

```
SQL> SELECT * FROM instructor WHERE DEPT_NAME LIKE 'c%';

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
501   Raju                 csm                       29001
502   Ramu                 csd                       30000
503   Ravi                 cse                       30000
504   Suresh               civil                     56000

SQL> |
```

13.Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	12

2.	Number of tables creation specified in observation	1
3.	Number of tables you created in the lab	1
4.	Number of Select Statements specified in the observation	12
5.	Number of Select statements you practised in lab	12
6	Number of Insert Statements specified in observation	7
7	Number of Insert Statements you practiced in Lab	7
8	Total number of Statements specified in lab	20
9	Total number of statements practiced in lab	20
10	Number of any addition statements practiced by you.	3

Experiment-6

23-NOV-2023

SQL queries to perform JOIN OPERATIONS

(CONDITIONAL JOIN, EQUI JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN)

1. Now create a table student with attributes name, rollno, branch.

```
SQL> CREATE TABLE student (
  2  ROLLNO NUMBER,
  3  NAME VARCHAR2(20),
  4  BRANCH VARCHAR2(20)
  5  );

Table created.

SQL>
```

2. Insert values into table student.

```
SQL> INSERT INTO student VALUES('11','Anu','cse');

1 row created.

SQL> INSERT INTO student VALUES('12','Mani','csm');

1 row created.

SQL> INSERT INTO student VALUES('13','Rani','csd');

1 row created.

SQL> INSERT INTO student VALUES('14','Vani','civil');

1 row created.

SQL> |
```

3. Using the command "SELECT * FROM student;" display table student.

```
SQL> SELECT * FROM student;

    ROLLNO NAME                 BRANCH
---------- -------------------- --------------------
        11 Anu                  cse
        12 Mani                 csm
        13 Rani                 csd
        14 Vani                 civil

SQL> |
```

4. Create a table library with attributes rollno, book.

```
SQL> CREATE TABLE library (
  2  ROLLNO NUMBER,
  3  BOOK VARCHAR2(20)
  4  );

Table created.

SQL>
```

5. Insert values into library table.

```
SQL> INSERT INTO library VALUES('12','DBMS');

1 row created.

SQL> INSERT INTO library VALUES('13','Java');

1 row created.

SQL> INSERT INTO library VALUES('14','Maths');

1 row created.

SQL>
```

6. Display table library using SELECT * FROM library command.

```
    ROLLNO NAME                 BRANCH               BOOK
---------- -------------------- -------------------- --------------------
        12 Mani                 csm                  DBMS
        13 Rani                 csd                  Java
        14 Vani                 civil                Maths

SQL>
```

7. CONDITIONAL JOIN: It helps in retrieving the desired data and performing complex queries.

```
SQL> SELECT * FROM student JOIN
  2  SELECT * FROM library WHERE student.ROLLNO=library.ROLLNO;
```

```
    ROLLNO NAME                 BRANCH               BOOK
---------- -------------------- -------------------- --------------------
        12 Mani                 csm                  DBMS
        13 Rani                 csd                  Java
```

8. EQUI JOIN: It helps in retrieving related information from different tables by matching corresponding values.

```
SQL> SELECT * FROM student JOIN library USING(ROLLNO);

    ROLLNO NAME                 BRANCH               BOOK
---------- -------------------- -------------------- --------------------
        12 Mani                 csm                  DBMS
        13 Rani                 csd                  Java
        14 Vani                 civil                Maths

SQL>
```

9. LEFT OUTER JOIN: It combines data from two or more tables based on the matching values in specified columns, but it also includes unmatched rows from the left table.

```
SQL> SELECT * FROM student NATURAL LEFT OUTER JOIN library;

    ROLLNO NAME                 BRANCH               BOOK
---------- -------------------- -------------------- --------------------
        12 Mani                 csm                  DBMS
        13 Rani                 csd                  Java
        14 Vani                 civil                Maths
        11 Anu                  cse

SQL>
```

10. RIGHT OUTER JOIN: It combines data from two or more tables based on the matching values in specified columns, but it also includes unmatched rows from the right table.

```
SQL> SELECT * FROM student NATURAL RIGHT OUTER JOIN library;

    ROLLNO NAME                 BRANCH               BOOK
---------- -------------------- -------------------- --------------------
        12 Mani                 csm                  DBMS
        13 Rani                 csd                  Java
        14 Vani                 civil                Maths

SQL> |
```

11. FULL OUTERJOIN: It includes all the rows from both the left and right tables, even if there is no match.

```
SQL> SELECT * FROM student NATURAL FULL OUTER JOIN library;

    ROLLNO NAME                 BRANCH               BOOK
---------- -------------------- -------------------- --------------------
        11 Anu                  cse
        12 Mani                 csm                  DBMS
        13 Rani                 csd                  Java
        14 Vani                 civil                Maths

SQL> |
```

12.Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	11
2.	Number of tables creation specified in observation	2
3.	Number of tables you created in the lab	2
4.	Number of Select Statements specified in the observation	7
5.	Number of Select statements you practised in lab	7
6	Number of Insert Statements specified in observation	7
7	Number of Insert Statements you practiced in Lab	7
8	Total number of Statements specified in lab	20
9	Total number of statements practiced in lab	20

Experiment-7

09-NOV-2023

SQL queries to perform AGGREGATE OPERATIONS (SUM, COUNT, AVG, MIN, MAX)

1. Now create a table Instructor with attributes id, name, salary.

```
SQL> CREATE TABLE instructor (
  2  ID VARCHAR2(5),
  3  NAME VARCHAR2(20) NOT NULL,
  4  DEPT_NAME VARCHAR2(20),
  5  SALARY NUMERIC(8,2)
  6  );

Table created.

SQL> |
```

2. Insert values into table instructor.

```
SQL> INSERT INTO instructor VALUES('101','Srinivas','com_sci','65000');

1 row created.

SQL> INSERT INTO instructor VALUES('102','Ravi','finance','90000');

1 row created.

SQL> INSERT INTO instructor VALUES('103','Ramu','music','40000');

1 row created.

SQL> INSERT INTO instructor VALUES('104','Ramesh','physics','95000');

1 row created.

SQL> INSERT INTO instructor VALUES('105','Suresh','com_sci','63000');

1 row created.

SQL> |
```

3. Using the command "SELECT * FROM instructor;" display table instructor.

```
SQL> SELECT * FROM INSTRUCTOR;

ID    NAME                 DEPT_NAME                SALARY
----- -------------------- -------------------- ----------
101   Srinivas             com_sci                   65000
102   Ravi                 finance                   90000
103   Ramu                 music                     40000
104   Ramesh               physics                   95000
105   Suresh               com_sci                   63000

SQL> |
```

4. Count: It displays the count of members present in instructor.

```
SQL> SELECT COUNT(*) FROM instructor;

  COUNT(*)
----------
         5

SQL>
```

5. AVERAGE(AVG): It displays average salary of each department.

```
SQL> SELECT DEPT_NAME,avg(salary) as avg_salary
  2  FROM instructor
  3  GROUP by DEPT_NAME;

DEPT_NAME            AVG_SALARY
-------------------- ----------
com_sci                   64000
finance                   90000
music                     40000
physics                   95000

SQL> |
```

6. Create a table student with attributes id, name, marks.

```
SQL> CREATE TABLE student (
  2  ID VARCHAR2(5),
  3  NAME VARCHAR2(20),
  4  MARKS NUMBER
  5  );

Table created.

SQL> |
```

7. Insert values into student table.

```
SQL> INSERT INTO student VALUES('501','Anu','20');

1 row created.

SQL> INSERT INTO student VALUES('502','Mani','30');

1 row created.

SQL> INSERT INTO student VALUES('503','Geetha','40');

1 row created.

SQL> |
```

8. Display table student using SELECT * FROM student command.

```
SQL> SELECT * FROM student;

ID    NAME                      MARKS
----- -------------------- ----------
501   Anu                          20
502   Mani                         30
503   Geetha                       40

SQL> |
```

9. SUM: It displays sum of all the marks from the table.

```
SQL> SELECT SUM(marks) FROM student;

SUM(MARKS)
----------
        90

SQL> |
```

10. MIN: It displays the minimum marks from the table.

```
SQL> SELECT MIN(marks) FROM student;

MIN(MARKS)
----------
        20

SQL> |
```

11. MAX: It displays the maximum marks from the table.

```
SQL> SELECT MAX(marks) FROM student;

MAX(MARKS)
----------
        40

SQL>
```

12.Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	11
2.	Number of tables creation specified in observation	2
3.	Number of tables you created in the lab	2
4.	Number of Select Statements specified in the observation	7
5.	Number of Select statements you practised in lab	7
6.	Number of Insert Statements specified in observation	8
7.	Number of Insert Statements you practiced in Lab	8
8.	Total number of Statements specified in lab	20
9.	Total number of statements practiced in lab	20

Experiment-8

23-NOV-2023

SQL queries to perform BUILT-IN FUNCTIONS (DATE, TIME)

1. LOWER (): It converts a string to lowercase.

```
SQL> SELECT LOWER('HELLO WORLD') FROM DUAL;

LOWER('HELL
-----------
hello world

SQL> |
```

2. UPPER (): It convers a string to uppercase.

```
SQL> SELECT UPPER('hello world') FROM DUAL;

UPPER('HELL
-----------
HELLO WORLD

SQL> |
```

3. INITCAP (): It returns the capitals of selected string.

```
INITCAP('HE
-----------
Hello World

SQL> |
```

4. CONCAT (): It adds two or more expressions together.

```
SQL> SELECT CONCAT('HELLO','WORLD') FROM DUAL;

CONCAT('HE
----------
HELLOWORLD

SQL>
```

5. SUBSTR (): It extracts a substring from a string.

```
SQL> SELECT SUBSTR('HELLO WORLD',1,5) FROM DUAL;

SUBST
-----
HELLO

SQL>
```

6. LENGTH (): It returns the length of the given string.

```
SQL> SELECT LENGTH('HELLO WORLD') FROM DUAL;

LENGTH('HELLOWORLD')
--------------------
                  11

SQL>
```

7. INSTR (): It returns the position or the first occurrence of a string in another string.

```
SQL> SELECT INSTR('HELLO WORLD','HELLO') FROM DUAL;

INSTR('HELLOWORLD','HELLO')
---------------------------
                          1

SQL>
```

8. TRIM (): It removes the selected one from string.

```
SQL> SELECT TRIM('H' FROM 'HELLO WORLD') FROM DUAL;

TRIM('H'FR
----------
ELLO WORLD

SQL>
```

9. Round (): It returns the specified values.

```
SQL> SELECT ROUND(45.626,2) FROM DUAL;

ROUND(45.626,2)
---------------
          45.63

SQL>
```

10. TRUNCATE (): It removes the decimal values which are specified.

```
SQL> SELECT TRUNC(45.626,0) FROM DUAL;

TRUNC(45.626,0)
---------------
             45

SQL>
```

11. MOD (): It returns the remainder.

```
SQL> SELECT MOD(1600,99) FROM DUAL;

MOD(1600,99)
------------
          16

SQL>
```

12. SYSDATE ():

```
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
---------
03-DEC-23

SQL>
```

13. MONTHS_BETWEEN ():

```
SQL> SELECT MONTHS_BETWEEN(SYSDATE,'17-NOV-23') FROM DUAL;

MONTHS_BETWEEN(SYSDATE,'17-NOV-23')
-----------------------------------
                        .562598193

SQL>
```

14. ADD_MONTHS ():

```
SQL> SELECT ADD_MONTHS(SYSDATE,5) FROM DUAL;

ADD_MONTH
---------
03-MAY-24

SQL>
```

15. NEXT_DAY ():

```
SQL> SELECT NEXT_DAY(SYSDATE,'WEDNESDAY') FROM DUAL;

NEXT_DAY(
---------
06-DEC-23

SQL> S
```

16. LAST_DAY ():

```
SQL> SELECT LAST_DAY(SYSDATE) FROM DUAL;

LAST_DAY(
---------
31-DEC-23

SQL>
```

17. TRUNC ():

```
SQL> SELECT TRUNC(SYSDATE,'DAY') FROM DUAL;

TRUNC(SYS
---------
03-DEC-23

SQL>
```

18. **Summary of the Lab Report.**

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	17
2.	Number of Select Statements specified in the observation	17
3.	Number of Select statements you practised in lab	17
4.	Total number of Statements specified in lab	21
5.	Total number of statements practiced in lab	21
6.	Number of any addition statements practiced by you.	3

Experiment-9

23-NOV-2023

SQL queries to perform KEY CONSTRAINTS (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, DEFAULT)

1. PRIMARY KEY: A primary key is a field which can uniquely identify each row in table and this constraint is used to specify a field as primary key.

```
SQL> CREATE  TABLE student (
  2  ID NUMBER,
  3  NAME VARCHAR(10),
  4  ADDRESS VARCHAR(20)
  5  );

Table created.

SQL>
```

2. FOREIGN KEY: A foreign key is a field which can uniquely identify each row in another table.

```
SQL> CREATE TABLE orders (
  2  O_ID NUMBER NOT NULL,
  3  C_ID NUMBER,
  4  PRIMARY KEY(O_ID),
  5  FOREIGN KEY(C_ID)REFERENCES customer(C_ID)
  6  );
```

```
Table created.
```

3. UNIQUE: This constraint when specified with a column, tells that the values in the column must be unique i.e, the values in any row of a column must not be repeated.

```
SQL> CREATE TABLE student (
  2  ID NUMBER UNIQUE,
  3  NAME VARCHAR(10),
  4  ADDRESS VARCHAR(20)
  5  );

Table created.

SQL>
```

4. NOT NULL: This constraint tells that we cannot store a null value in a column.

```
SQL> CREATE TABLE student (
  2  ID NUMBER,
  3  NAME VARCHAR(10) NOT NULL,
  4  ADDRESS VARCHAR(20)
  5  );

Table created.

SQL> |
```

5. DEFAULT: This constraint specifies a default value for the column when no value is specified by the user.

```
SQL> CREATE TABLE student (
  2  ID NUMBER,
  3  NAME VARCHAR(10) NOT NULL,
  4  AGE NUMBER DEFAULT 18
  5  );

Table created.

SQL> |
```

6. CHECK: This constraint helps to validate the value for the column to meet a particular condition i.e it helps to ensure that the value stored in a column meets a specific condition.

```
SQL> CREATE TABLE student (
  2  ID NUMBER NOT NULL,
  3  NAME VARCHAR(10) NOT NULL,
  4  AGE NUMBER NOT NULL CHECK(AGE>=18)
  5  );

Table created.

SQL> |
```

7. **Summary of the Lab Report.**

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	6
2.	Number of tables creation specified in observation	6
3.	Number of tables you created in the lab	6
4.	Total number of Statements specified in lab	20
5.	Total number of statements practiced in lab	20
6.	Number of any addition statements practiced by you.	3

Experiment-10

30-NOV-2023

PL/SQL program for calculating the factorial of a given number

1. Now write the code to find factorial of given number using WHILE LOOP. We use '/' to end and execute the program.

```
C:\Windows\system32\cmd.e:

SQL> DECLARE
  2  fact NUMBER:=1;
  3  n NUMBER;
  4  n1 NUMBER;
  5  BEGIN
  6  n:=&n;
  7  n1:=n;
  8  WHILE n>0 LOOP
  9  fact:=n*fact;
 10  n:=n-1;
 11  END LOOP;
 12  DBMS_OUTPUT.PUT_LINE('The Factorial of '||n1||' is '||fact);
 13  END;
 14  /
```

2. After the execution the following displays.

```
Enter value for n: 7
old   6: n:=&n;
new   6: n:=7;

PL/SQL procedure successfully completed.
```

3. To display the output, we use SET SERVEROUT ON. The following output is displayed giving the factorial of the given number.

```
PL/SQL procedure successfully completed.

SQL> SET SERVEROUT ON
SQL> /
Enter value for n: 7
old   6: n:=&n;
new   6: n:=7;
The Factorial of 7 is 5040

PL/SQL procedure successfully completed.

SQL> |
```

4. **Summary of the Lab Report.**

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	3

2.	Number of tables creation specified in observation	3
3.	Number of Select statements you practised in lab	5
4.	Total number of programs specified in lab	1
5.	Total number of programs practiced in lab	1

Experiment-11

30-NOV-23

PL/SQL program for finding the given number is prime or not

1. Now write the code to find given number is prime or not. We use '/' to end and execute the program.

```
C:\Windows\system32\cmd.e:

  1  DECLARE
  2  n NUMBER;
  3  n1 NUMBER;
  4  i NUMBER;
  5  temp NUMBER;
  6  BEGIN
  7  n :=&n;
  8  n1:=n;
  9  i := 2;
 10  temp := 1;
 11  FOR i IN 2..n/2
 12  LOOP
 13  IF MOD(n, i) = 0
 14  THEN
 15  temp := 0;
 16  EXIT;
 17  END IF;
 18  END LOOP;
 19  IF temp = 1
 20  THEN
 21  DBMS_OUTPUT.PUT_LINE(n||' is a prime number');
 22  ELSE
 23  DBMS_OUTPUT.PUT_LINE(n||' is not a prime number');
 24  END IF;
 25* END;
SQL> /
```

2. After the execution the following displays.

```
Enter value for n: 543
old   7: n :=&n;
new   7: n :=543;

PL/SQL procedure successfully completed.
```

3. To display the output, we use SET SERVEROUT ON. The following output is displayed that given number is prime or not.

```
SQL> SET SERVEROUT ON
SQL> /
Enter value for n: 543
old   7: n :=&n;
new   7: n :=543;
543 is not a prime number

PL/SQL procedure successfully completed.

SQL> |
```

4. Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	3
2.	Number of tables creation specified in observation	3
3.	Number of Select statements you practised in lab	5
4.	Total number of programs specified in lab	1
5.	Total number of programs practiced in lab	1

Experiment-12

30-11-23

PL/SQL program for displaying the Fibonacci series up to an integer

1. Now write the code to display Fibonacci series until the given number. We use '/' to end and execute the program.

```
C:\Windows\system32\cmd.e:

SQL> DECLARE
  2  FIRST NUMBER := 0;
  3  SECOND NUMBER := 1;
  4  TEMP NUMBER;
  5  N NUMBER;
  6  N1 NUMBER;
  7  I NUMBER;
  8  BEGIN
  9  N:=&N;
 10  N1:=N;
 11  DBMS_OUTPUT.PUT_LINE('SERIES:');
 12  DBMS_OUTPUT.PUT_LINE(FIRST);
 13  DBMS_OUTPUT.PUT_LINE(SECOND);
 14  FOR I IN 2..N
 15  LOOP
 16  TEMP:=FIRST+SECOND;
 17  FIRST := SECOND;
 18  SECOND := TEMP;
 19  DBMS_OUTPUT.PUT_LINE(TEMP);
 20  END LOOP;
 21  END;
 22  /
```

2. After the execution the following displays.

```
Enter value for n: 9
old   9: N:=&N;
new   9: N:=9;

PL/SQL procedure successfully completed.
```

3. To display the output, we use SET SERVEROUT ON. The following output is displaying the Fibonacci series until the given number.

```
SQL> SET SERVEROUT ON
SQL> /
Enter value for n: 9
old   9: N:=&N;
new   9: N:=9;
SERIES:
0
1
1
2
3
5
8
13
21
34

PL/SQL procedure successfully completed.

SQL> |
```

4. Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	3
2.	Number of tables creation specified in observation	3
3.	Number of Select statements you practised in lab	5
4.	Total number of programs specified in lab	1
5.	Total number of programs practiced in lab	1

Experiment-13

07-DEC-2023

PL/SQL program to implement Stored Procedure on table

1. Now create a table sailor1 with attributes id, name.

```
SQL> CREATE TABLE sailor1(
  2       id NUMBER PRIMARY KEY,
  3       name VARCHAR2(50) NOT NULL
  4  );

Table created.

SQL>
```

2. Create procedure.

```
SQL> CREATE OR REPLACE PROCEDURE insertuser(id IN NUMBER,name IN VARCHAR2)
  2  AS
  3  BEGIN
  4  INSERT INTO sailor1 VALUES(id,name);
  5  DBMS_OUTPUT.PUT_LINE('Record inserted successfully');
  6  END;
  7  /

Procedure created.

SQL>
```

3. Insert a user into the table.

```
SQL> DECLARE
  2  co NUMBER;
  3  BEGIN
  4  insertuser(11,'RANI');
  5  SELECT COUNT(*) INTO co FROM sailor1;
  6  DBMS_OUTPUT.PUT_LINE(co||' Record is inserted successfully');
  7  END;
  8  /

PL/SQL procedure successfully completed.

SQL>
```

4. Insert two records into the table.

```
SQL> DECLARE
  2  co NUMBER;
  3  BEGIN
  4  insertuser(43,'Madhuri');
  5  SELECT COUNT(*) INTO co FROM sailor1;
  6  DBMS_OUTPUT.PUT_LINE(co||' Record is inserted successfully');
  7  END;
  8  /
Record inserted successfully
2 Record is inserted successfully

PL/SQL procedure successfully completed.

SQL> |
```

5. Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	4
2.	Number of tables creation specified in observation	1
3.	Number of tables you created in the lab	1
4.	Number of Insert Statements specified in observation	2
5.	Number of Insert Statements you practiced in Lab	2
6.	Total number of Statements specified in lab	5
7.	Total number of statements practiced in lab	5

Experiment-14

07-DEC-2023

PL/SQL program to implement Stored Function on table

1. Now create a table section with attributes id, course_name,strength.

```
SQL> CREATE TABLE section(
  2  id NUMBER PRIMARY KEY,
  3  course_name VARCHAR2(20) NOT NULL,
  4  strength NUMBER NOT NULL
  5  );

Table created.

SQL>
```

2. Insert rows into Section table.

```
SQL> INSERT ALL
  2  INTO section VALUES (1,'CSE',50)
  3  INTO section VALUES (2,'CSM',60)
  4  INTO section VALUES (3,'CSD',75)
  5  SELECT * FROM dual;

3 rows created.

SQL> |
```

3. Create a function.

```
SQL> CREATE OR REPLACE FUNCTION totalstrength RETURN NUMBER
  2  AS
  3  total NUMBER:=0;
  4  BEGIN
  5  SELECT sum(strength) INTO total FROM section;
  6  return total;
  7  END;
  8  /

Function created.

SQL> |
```

4. Displaying the strength of students.

```
SQL> DECLARE
  2  answer NUMBER;
  3  BEGIN
  4  answer:=totalstrength();
  5  DBMS_OUTPUT.PUT_LINE('Total strength of students is '||answer);
  6  END;
  7  /
Total strength of students is 185

PL/SQL procedure successfully completed.

SQL>
```

5. **Summary of the Lab Report.**

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	8
2.	Number of tables creation specified in observation	1
3.	Number of tables you created in the lab	1
4.	Number of Insert Statements specified in observation	2
5.	Number of Insert Statements you practiced in Lab	2
6.	Total number of Statements specified in lab	5
7.	Total number of statements practiced in lab	5

Experiment-15

07-DEC-2023

PL/SQL program to implement Trigger on table

1. Now create a table instruc with attributes id, name,dept_name.

```
SQL> CREATE TABLE instruc(
  2  id NUMBER PRIMARY KEY,
  3  name VARCHAR2(50) NOT NULL,
  4  dept_name VARCHAR2(20) NOT NULL,
  5  salary NUMBER(10,2) CHECK(salary>10000)
  6  );

Table created.
```

2. Insert values into table instruc.

```
SQL> INSERT ALL
  2  INTO instruc VALUES (1,'Abhi','CSE',50000)
  3  INTO instruc VALUES (2,'Narsimha','CSM',75000)
  4  INTO instruc VALUES (3,'Balaji','CSE',80000)
  5  INTO instruc VALUES (4,'Rani','CSD',47000)
  6  SELECT * FROM dual;

4 rows created.

SQL>
```

3. Create a Trigger.

```
SQL> CREATE OR REPLACE TRIGGER display_changes
  2  BEFORE UPDATE ON instruc
  3  FOR EACH ROW
  4  WHEN (NEW.ID = OLD.ID)
  5  DECLARE
  6  sal_diff number;
  7  BEGIN
  8  sal_diff := :NEW.salary - :OLD.salary;
  9  dbms_output.put_line('Old salary: ' || :OLD.salary);
 10  dbms_output.put_line('New salary: ' || :NEW.salary);
 11  dbms_output.put_line('Salary difference: ' || sal_diff);
 12  END;
 13  /

Trigger created.

SQL>
```

4. Updating the values of the rows by the given condition.

```
C:\Windows\system32\cmd.e:

SQL> DECLARE
  2  tot_rows NUMBER;
  3  BEGIN
  4  UPDATE instruc
  5  SET salary=salary*1.5;
  6  IF sql%notfound THEN
  7  DBMS_OUTPUT.PUT_LINE('no instructors updated');
  8  ELSIF sql%found THEN
  9  tot_rows:=sql%rowcount;
 10  DBMS_OUTPUT.PUT_LINE(tot_rows||' instructors updated');
 11  END IF;
 12  END;
 13  /
```

5. The values are updated.

```
Old salary: 50000
New salary: 75000
Salary difference: 25000
Old salary: 75000
New salary: 112500
Salary difference: 37500
Old salary: 80000
New salary: 120000
Salary difference: 40000
Old salary: 47000
New salary: 70500
Salary difference: 23500
4 instructors updated

PL/SQL procedure successfully completed.

SQL>
```

6. Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	5
2.	Number of tables creation specified in observation	1
3.	Number of tables you created in the lab	1
4.	Number of Insert Statements specified in observation	2
5.	Number of Insert Statements you practiced in Lab	2
6.	Total number of Statements specified in lab	5
7.	Total number of statements practiced in lab	5

Experiment-16

07-DEC-2023

PL/SQL program to implement Cursor on table

1. Now create a table customers with attributes id, name, age, salary,.

```
SQL> CREATE TABLE customers(
  2  id NUMBER PRIMARY KEY,
  3  name VARCHAR2(30) NOT NULL,
  4  age NUMBER(3) NOT NULL,
  5  salary NUMBER(10,2) NOT NULL
  6  );

Table created.

SQL> |
```

2. Updating the rows.

```
SQL> DECLARE
  2  tot_rows NUMBER;
  3  BEGIN
  4  UPDATE customers SET salary=salary*1.5;
  5  IF sql%notfound THEN
  6  DBMS_OUTPUT.PUT_LINE('No customers updated');
  7  ELSIF sql%found THEN
  8  tot_rows := sql%rowcount;
  9  DBMS_OUTPUT.PUT_LINE(tot_rows||' customers updated');
 10  END IF;
 11  END;
 12  /
No customers updated

PL/SQL procedure successfully completed.

SQL> |
```

3. Inserting values into customer table.

```
SQL> INSERT ALL
  2  INTO customers VALUES (501,'Ramu',22,60000)
  3  INTO customers VALUES (502,'Ramesh',33,70000)
  4  INTO customers VALUES (503,'Suresh',23,65000)
  5  INTO customers VALUES (504,'Ravi',25,60000)
  6  SELECT * FROM dual;

4 rows created.

SQL>
```

4. Updating the values of the rows by the given condition.

```
C:\Windows\system32\cmd.e:

SQL> DECLARE
  2  tot_rows NUMBER;
  3  BEGIN
  4  UPDATE instruc
  5  SET salary=salary*1.5;
  6  IF sql%notfound THEN
  7  DBMS_OUTPUT.PUT_LINE('no instructors updated');
  8  ELSIF sql%found THEN
  9  tot_rows:=sql%rowcount;
 10  DBMS_OUTPUT.PUT_LINE(tot_rows||' instructors updated');
 11  END IF;
 12  END;
 13  /
```

5. Program using explicit cursor.

s

```
SQL> DECLARE
  2  c_id customers.id%type;
  3  c_name customers.name%type;
  4  c_age customers.age%type;
  5  CURSOR c_customers IS
  6  SELECT id,name,age FROM customers;
  7  BEGIN
  8  OPEN c_customers;
  9  LOOP
 10  FETCH c_customers INTO c_id,c_name,c_age;
 11  EXIT WHEN c_customers%notfound;
 12  DBMS_OUTPUT.PUT_LINE(c_id||' '||c_name||' '||c_age);
 13  END LOOP;
 14  CLOSE c_customers;
 15  END;
 16  /
501 Ramu 22
502 Ramesh 33
503 Suresh 23
504 Ravi 25

PL/SQL procedure successfully completed.

SQL>
```

6. Summary of the Lab Report.

SNO	Summary Information	Total
1.	Number of Screen Shorts taken	5
2.	Number of tables creation specified in observation	1
3.	Number of tables you created in the lab	1
4.	Number of Insert Statements specified in observation	2
5.	Number of Insert Statements you practiced in Lab	2
6.	Total number of Statements specified in lab	5
7.	Total number of statements practiced in lab	5

- The Overall experiments report as follows.

Summary of the Lab Report

SNO	Summary Information	Total
1.	Total Number of Screen Shorts taken	143
2.	Number of tables creation specified in observation	32
3.	Number of tables you created in the lab	37
4.	Number of Select Statements specified in the observation	85
5.	Number of Select statements you practised in lab	97
6	Number of Insert Statements specified in observation	47
7	Number of Insert Statements you practiced in Lab	54
8.	Number of Alter Statements specified in observation	10
9.	Number of Alter Statements practiced in lab	10
10.	Number of views created in observation	2
11.	Number of views practiced in lab	2
10	Total number of Statements specified in observation	170
11	Total number of statements practiced in lab	200
12	Number of any addition statements practiced by you.	12