Q. 1. Implementation of DDL commands of SQL with suitable example▪ Create table [20 Marks]

▪ Alter table

▪ Drop Table

Create table student(Roll no ,sname,date of birth).Add new column into student

relation name address as text data type and column phone of data type integer

```
CREATE TABLE student (
   RollNo INT PRIMARY KEY,
   sname VARCHAR(50),
   dateOfBirth DATE
);


ALTER TABLE student
ADD address TEXT,
ADD phone INT;


DROP TABLE student;
```

//

Q. 2. Write a PL/SQL Program to Find Factorial of a Number

```
SET SERVEROUTPUT ON;

-- PL/SQL program to find factorial

DECLARE

   num NUMBER := 5; -- Change this value to find factorial for a different number

   result NUMBER := 1;

BEGIN

   FOR i IN 1..num LOOP

      result := result * i;

   END LOOP;


   DBMS_OUTPUT.PUT_LINE('Factorial of ' || num || ' is ' || result);

END;

/
```

Factorial of 5 is: 120

Q. 1. Implementation of different types of function with suitable examples

- By using Number function
- By using Aggregate Function
- By using Character Function
- By using Conversion Function
- By using Date Function

By using Number function

```
CREATE TABLE sales (
   amount NUMERIC(105.27 ,33.84 ,72.91 );
2)
SELECT ROUND(amount, 2) AS rounded_amount
FROM sales;
```

By using Aggregate Function:

```
SELECT SUM(amount) AS total_sales
FROM sales;
```

By using Character Function:

```
CREATE TABLE employees (
   first_name VARCHAR(50)
);
```

```
INSERT INTO employees (first_name) VALUES ('John');
INSERT INTO employees (first_name) VALUES ('Alice');
INSERT INTO employees (first_name) VALUES ('Bob');
```

```
SELECT UPPER(first_name) AS upper_first_name
FROM employees;
```

By using Conversion Function:

```
CREATE TABLE orders (
   order_date DATE
);

INSERT INTO orders (order_date) VALUES ('2023-07-10');
INSERT INTO orders (order_date) VALUES ('2023-07-15');
INSERT INTO orders (order_date) VALUES ('2023-07-20');
```

We can use the TO_CHAR function to convert the dates to a specific date format:

```
SELECT TO_CHAR(order_date, 'DD-MON-YYYY') AS formatted_date
FROM orders;
```

Result

formatted_date

10-JUL-2023

15-JUL-2023

20-JUL-2023

By using Date Function

Example:

```
SELECT SYSDATE AS current_date_time
FROM dual;
```

Q. 1. PL/SQL Program to Print Table of a Number [20 Marks]

```
SET SERVEROUTPUT ON;

-- PL/SQL program to print the table of a number

DECLARE

   num NUMBER := 5; -- Change this value to print the table for a different number

BEGIN

   DBMS_OUTPUT.PUT_LINE('Table of ' || num);

   DBMS_OUTPUT.PUT_LINE('----------------');

   FOR i IN 1..10 LOOP

      DBMS_OUTPUT.PUT_LINE(num || ' * ' || i || ' = ' || num * i);

   END LOOP;

END;

/

.....

BEGIN

   print_table(5); -- Replace 5 with any number for which you want to print the table

END;

/
```

Q. 2. Implementation of different types of Joins

- Inner Join
- Outer Join
- Natural Join [20 Marks]

Customer(customer_id,first_name)

Order(order_id,amount)

Cust_order(customerid , first_amount,amount)

```
CREATE TABLE Customer (
   customer_id INT,
   first_name VARCHAR(50)
);
INSERT INTO Customer (customer_id, first_name) VALUES (1, 'John');
INSERT INTO Customer (customer_id, first_name) VALUES (2, 'Alice');
INSERT INTO Customer (customer_id, first_name) VALUES (3, 'Bob');


CREATE TABLE "Order" (
   order_id INT,
   amount NUMERIC(10, 2)
);
INSERT INTO "Order" (order_id, amount) VALUES (1001, 500);
INSERT INTO "Order" (order_id, amount) VALUES (1002, 300);
INSERT INTO "Order" (order_id, amount) VALUES (1003, 200);

CREATE TABLE Cust_order (
   customer_id INT,
   first_amount NUMERIC(10, 2),
```

```
    amount NUMERIC(10, 2)
);
```

Inner Join:

```
SELECT c.customer_id, c.first_name, co.first_amount, co.amount
FROM Customer c
INNER JOIN Cust_order co ON c.customer_id = co.customer_id;
```

result of innere join

ustomer_id	first_name	first_amount	amount
1	John	100	500
2	Alice	200	300
3	Bob	300	200

Left Outer Join:

```
SELECT c.customer_id, c.first_name, co.first_amount, co.amount
FROM Customer c
LEFT JOIN Cust_order co ON c.customer_id = co.customer_id;
```

Result of Left Outer Join:

customer_id	first_name	first_amount	amount
1	John	100	500
2	Alice	200	300
3	Bob	300	200

NATURAL join

```
SELECT customer_id, first_name, first_amount, amount
FROM Customer NATURAL JOIN Cust_order;
```

Result of Natural Join:

customer_id	first_name	first_amount	amount
1	John	100	500
2	Alice	200	300
3	Bob	300	200

Q. 2. Write a program to implement SQL Cursors.

```
CREATE TABLE employees (
   employee_id INT PRIMARY KEY,
   first_name VARCHAR(50),
   last_name VARCHAR(50),
   department VARCHAR(50),
   salary NUMERIC(10, 2)
);
```

.....

```
SET SERVEROUTPUT ON;

-- PL/SQL program with a cursor
```

```
DECLARE
   -- Declare variables to store data from the cursor
   emp_id employees.employee_id%TYPE;
   emp_first_name employees.first_name%TYPE;
   emp_last_name employees.last_name%TYPE;
   emp_department employees.department%TYPE;
   emp_salary employees.salary%TYPE;

   -- Declare the cursor
   CURSOR emp_cursor IS
      SELECT employee_id, first_name, last_name, department, salary
      FROM employees;

BEGIN
   -- Open the cursor
   OPEN emp_cursor;

   -- Fetch data from the cursor and process it
   LOOP
      FETCH emp_cursor INTO emp_id, emp_first_name, emp_last_name, emp_department, emp_salary;

      -- Exit the loop if there is no more data to fetch
      EXIT WHEN emp_cursor%NOTFOUND;

      -- Process the fetched data (print it in this example)
```

```
        DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_id);
        DBMS_OUTPUT.PUT_LINE('Name: ' || emp_first_name || ' ' || emp_last_name);
        DBMS_OUTPUT.PUT_LINE('Department: ' || emp_department);
        DBMS_OUTPUT.PUT_LINE('Salary: ' || emp_salary);
        DBMS_OUTPUT.PUT_LINE('--------------------------');
    END LOOP;

    -- Close the cursor
    CLOSE emp_cursor;
END;
/
```

//

Q)Study & Implementation of SQL Triggers.

INSERT Trigger:

An INSERT trigger is executed automatically after an INSERT operation on the table. Let's create an INSERT trigger that inserts a record into an "order_log" table whenever a new order is inserted into the "orders" table.

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    order_date DATE,
    amount NUMERIC(10, 2)
);



CREATE TABLE order_log (
```

```
    log_id INT PRIMARY KEY,
    order_id INT,
    action_date DATE
);

CREATE OR REPLACE TRIGGER orders_insert_trigger
AFTER INSERT ON orders
FOR EACH ROW
BEGIN
    INSERT INTO order_log (log_id, order_id, action_date)
    VALUES (NULL, :NEW.order_id, SYSDATE);
END;
/.....................................................
//or update trigger
ALTER TABLE orders ADD last_modified DATE;

CREATE OR REPLACE TRIGGER orders_update_trigger
AFTER UPDATE OF amount ON orders
FOR EACH ROW
BEGIN
    UPDATE orders
    SET last_modified = SYSDATE
    WHERE order_id = :NEW.order_id;
END;
/
```

//

Q. 2. Implementation of different types of operators in SQL

- By using Arithmetic Operators
- By using Logical Operators
- By using Comparison Operator
- By using Special Operator
- By using Set Operation

- By using Arithmetic Operators

```
CREATE TABLE numbers (
   a INT,
   b INT
);

INSERT INTO numbers (a, b) VALUES (10, 5);
INSERT INTO numbers (a, b) VALUES (20, 8);
INSERT INTO numbers (a, b) VALUES (7, 3);
-- Assuming we have a table called "numbers" with two columns "a" and "b"
SELECT a, b, a + b AS sum, a - b AS difference, a * b AS product, a / b AS division, a % b AS modulus
FROM numbers;
```

- By using Logical Operators

```
CREATE TABLE students (
   name VARCHAR(50),
   age INT
```

```
);

INSERT INTO students (name, age) VALUES ('John', 20);
INSERT INTO students (name, age) VALUES ('Alice', 22);
INSERT INTO students (name, age) VALUES ('Bob', 25);
INSERT INTO students (name, age) VALUES ('Emma', 18);
INSERT INTO students (name, age) VALUES ('Michael', 27);

-- Assuming we have a table called "students" with columns "name" and "age"
SELECT name, age
FROM students
WHERE age > 18 AND age < 25;
```

By using Comparison Operator

```
CREATE TABLE employees (
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    salary NUMERIC(10, 2),
    department VARCHAR(50)
);

INSERT INTO employees (first_name, last_name, salary, department) VALUES ('John', 'Doe', 55000.00, 'IT');
INSERT INTO employees (first_name, last_name, salary, department) VALUES ('Alice', 'Smith', 60000.00, 'HR');
```

```
INSERT INTO employees (first_name, last_name, salary, department) VALUES ('Bob', 'Johnson', 48000.00, 'Finance');

INSERT INTO employees (first_name, last_name, salary, department) VALUES ('Emma', 'Lee', 70000.00, 'IT');

INSERT INTO employees (first_name, last_name, salary, department) VALUES ('Michael', 'Brown', 52000.00, 'Finance');

-- Assuming we have a table called "employees" with columns "salary" and "department"

SELECT *

FROM employees

WHERE salary >= 50000 AND department = 'IT';
```

Special Operators:

```
CREATE TABLE employees (

   first_name VARCHAR(50),

   last_name VARCHAR(50)

);


INSERT INTO employees (first_name, last_name) VALUES ('John', 'Doe');

INSERT INTO employees (first_name, last_name) VALUES ('Alice', 'Smith');

INSERT INTO employees (first_name, last_name) VALUES ('Bob', 'Johnson');

INSERT INTO employees (first_name, last_name) VALUES ('Emma', 'Lee');

INSERT INTO employees (first_name, last_name) VALUES ('Michael', 'Brown');

-- Assuming we have a table called "employees" with columns "first_name" and "last_name"

SELECT first_name || ' ' || last_name AS full_name

FROM employees

WHERE last_name IS NOT NULL;
```

Set Operation:set operations are used to combine the results of two or more SELECT queries.

```
CREATE TABLE table1 (
   id INT,
   value VARCHAR(50)
);

CREATE TABLE table2 (
   id INT,
   value VARCHAR(50)
);

-- Insert data into table1
INSERT INTO table1 (id, value) VALUES (1, 'Value 1');
INSERT INTO table1 (id, value) VALUES (2, 'Value 2');
INSERT INTO table1 (id, value) VALUES (3, 'Value 3');

-- Insert data into table2
INSERT INTO table2 (id, value) VALUES (2, 'Value 2');
INSERT INTO table2 (id, value) VALUES (3, 'Value 3');
INSERT INTO table2 (id, value) VALUES (4, 'Value 4');
-- Assuming we have two tables "table1" and "table2" with the same columns "id" and "value"
-- UNION operation to combine the results and remove duplicates
SELECT id, value FROM table1
UNION
SELECT id, value FROM table2;
```