EX NO:1 CREATE A DATABASE TABLE AND MANIPULATE DATA USING SQL DDL AND DML COMMANDS

QUERY:

```
CREATE TABLE students (
  id INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  age INTEGER CHECK (age >= 18),
  email VARCHAR(255) UNIQUE,
  gender CHAR(10) NOT NULL
);
Query OK, 0 rows affected (0.34 sec)
-- Insert rows
INSERT INTO students (id, name, age, email, gender)
VALUES
  (1, 'Prakash', 20, 'prakash@email.com', 'Male'), (2, 'Rose', 22, 'rose@email.com', 'Female');
Query OK, 2 rows affected (0.13 sec)
Records: 2  Duplicates: 0  Warnings: 0


Select*from students;
+----+---------+------+-------------------+--------+
| id | name    | age  | email             | gender |

+----+---------+------+-------------------+--------+

|  1 | Prakash |   20 | prakash@email.com | Male   |

|  2 | Rose    |   22 | rose@email.com    | Female |

+----+---------+------+-------------------+--------+

2 rows in set (0.00 sec)
-- Update a row
UPDATE students
SET age = 23
WHERE id = 2;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0
Select*from students;
+----+---------+------+-------------------+--------+

| id | name    | age  | email             | gender |
```

```
+----+---------+------+------------------+--------+

|  1 | Prakash |   20 | prakash@email.com | Male   |

|  2 | Rose    |   23 | rose@email.com    | Female |

+----+---------+------+------------------+--------+

2 rows in set (0.00 sec)
-- Delete a row
DELETE FROM students
WHERE id = 2;
Query OK, 1 row affected (0.09 sec)
Select*from students;
+----+---------+------+------------------+--------+
| id | name    | age  | email            | gender |

+----+---------+------+------------------+--------+

|  1 | Prakash |   20 | prakash@email.com | Male   |

+----+---------+------+------------------+--------+

1 row in set (0.00 sec)
```

EX NO:2 CREATE A SET OF TABLES, ADD FOREIGN KEY CONSTRAINTS AND INCORPORATE REFERENTIAL INTEGRITY

```
CREATE TABLE departments (  department_id INT PRIMARY KEY,
   department_name VARCHAR(50) NOT NULL
);
Query OK, 0 rows affected (0.16 sec)
CREATE TABLE employees (
   employee_id INT PRIMARY KEY,
   employee_name VARCHAR(50) NOT NULL,
   department_id INT,
   salary DECIMAL(10,2),
   FOREIGN KEY (department_id) REFERENCES departments (department_id)
   ON DELETE RESTRICT ON UPDATE CASCADE
);
Query OK, 0 rows affected (0.19 sec)
INSERT INTO departments (department_id, department_name)
VALUES (1, 'Sales'), (2, 'Marketing'), (3, 'IT'), (4, 'Finance');
SELECT*FROM DEPARTMENTS;
+---------------+-----------------+
| department_id | department_name |
+---------------+-----------------+
|             1 | Sales           |
|             2 | Marketing       |
|             3 | IT              |
|             4 | Finance         |
+---------------+-----------------+
```

4 rows in set (0.00 sec)

```
INSERT INTO employees (employee_id, employee_name, department_id, salary)
VALUES (1, 'John Doe', 1, 50000), (2, 'Jane Doe', 2, 55000),
    (3, 'Jim Smith', 3, 60000), (4, 'Sarah Johnson', 4, 65000),
    (5, 'Michael Johnson', 1, 55000), (6, 'Emily Davis', 2, 52000),
    (7, 'David Brown', 3, 60000), (8, 'Katie Wilson', 4, 63000),
    (9, 'William Davis', 1, 55000), (10, 'Emily Wilson', 2, 53000),
    (11, 'James Brown', 3, 65000);
SELECT*FROM EMPLOYEES;
+-------------+-----------------+---------------+----------+
| employee_id | employee_name   | department_id | salary   |
+-------------+-----------------+---------------+----------+
|           1 | John Doe        |             1 | 50000.00 |
|           2 | Jane Doe        |             2 | 55000.00 |
|           3 | Jim Smith       |             3 | 60000.00 |
|           4 | Sarah Johnson   |             4 | 65000.00 |
|           5 | Michael Johnson |             1 | 55000.00 |
|           6 | Emily Davis     |             2 | 52000.00 |
|           7 | David Brown     |             3 | 60000.00 |
|           8 | Katie Wilson    |             4 | 63000.00 |
|           9 | William Davis   |             1 | 55000.00 |
|          10 | Emily Wilson    |             2 | 53000.00 |
|          11 | James Brown     |             3 | 65000.00 |
+-------------+-----------------+---------------+----------+
```

20 rows in set (0.00 sec)

EX NO:3 QUERY DATABASE TABLES USING WHERE CLAUSE CONDITIONS AND AGGREGATE FUNCTIONS

```
CREATE TABLE departments (  department_id INT PRIMARY KEY,
   department_name VARCHAR(50) NOT NULL
);
Query OK, 0 rows affected (0.16 sec)
CREATE TABLE employees (
   employee_id INT PRIMARY KEY,
   employee_name VARCHAR(50) NOT NULL,
   department_id INT,
   salary DECIMAL(10,2),
   FOREIGN KEY (department_id) REFERENCES departments (department_id)
   ON DELETE RESTRICT ON UPDATE CASCADE
);
Query OK, 0 rows affected (0.19 sec)
INSERT INTO departments (department_id, department_name)
VALUES (1, 'Sales'), (2, 'Marketing'), (3, 'IT'), (4, 'Finance');
SELECT*FROM DEPARTMENTS;
+---------------+-----------------+
| department_id | department_name |
+---------------+-----------------+
|             1 | Sales           |
|             2 | Marketing       |
|             3 | IT              |
|             4 | Finance         |
+---------------+-----------------+
```

4 rows in set (0.00 sec)

```
INSERT INTO employees (employee_id, employee_name, department_id, salary)
VALUES (1, 'John Doe', 1, 50000), (2, 'Jane Doe', 2, 55000),
      (3, 'Jim Smith', 3, 60000), (4, 'Sarah Johnson', 4, 65000),
      (5, 'Michael Johnson', 1, 55000);
SELECT*FROM EMPLOYEES;
+-------------+-----------------+---------------+----------+
| employee_id | employee_name   | department_id | salary   |
+-------------+-----------------+---------------+----------+
|           1 | John Doe        |             1 | 50000.00 |
|           2 | Jane Doe        |             2 | 55000.00 |
|           3 | Jim Smith       |             3 | 60000.00 |
|           4 | Sarah Johnson   |             4 | 65000.00 |
|           5 | Michael Johnson |             1 | 55000.00 |
+-------------+-----------------+---------------+----------+
20 rows in set (0.00 sec)
-- Select all employees who work in a specific department
SELECT * FROM employees
WHERE department_id = 1;

+-------------+-----------------+---------------+----------+
| employee_id | employee_name   | department_id | salary   |
+-------------+-----------------+---------------+----------+
|           1 | John Doe        |             1 | 50000.00 |
|           5 | Michael Johnson |             1 | 55000.00 |
+-------------+-----------------+---------------+----------+
2 rows in set (0.00 sec)
```

```
-- Select employee names and department names from both tables using a join
SELECT employees.employee_name, departments.department_name
FROM employees
JOIN departments
ON employees.department_id = departments.department_id;
+-----------------+-----------------+
| employee_name   | department_name |
+-----------------+-----------------+
| John Doe        | Sales           |
| Michael Johnson | Sales           |
| Jane Doe        | Marketing       |
| Jim Smith       | IT              |
| Sarah Johnson   | Finance         |
+-----------------+-----------------+
5 rows in set (0.05 sec)
-- Select the average salary of employees in each department
SELECT departments.department_name, AVG(salary)
FROM employees
JOIN departments
ON employees.department_id = departments.department_id
GROUP BY departments.department_name;
+-----------------+--------------+
| department_name | AVG(salary)  |
+-----------------+--------------+
| Finance         | 65000.000000 |
| IT              | 60000.000000 |
| Marketing       | 55000.000000 |
| Sales           | 52500.000000 |
```

+----------------+--------------+

4 rows in set (0.03 sec)

EX NO:4 QUERY DATABASE TABLES USING SUBQUERIES AND SIMPLE JOIN OPERATIONS

```
CREATE TABLE departments (
    department_id INT PRIMARY KEY,
    department_name VARCHAR(50) NOT NULL
);
Query OK, 0 rows affected (0.13 sec)
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(50) NOT NULL,
    department_id INT,
    salary DECIMAL(10,2),
    FOREIGN KEY (department_id) REFERENCES departments (department_id)
    ON DELETE RESTRICT ON UPDATE CASCADE
);
Query OK, 0 rows affected (0.14 sec)
INSERT INTO departments (department_id, department_name)
VALUES (1, 'Sales'), (2, 'Marketing'), (3, 'IT'), (4, 'Finance');
Query OK, 4 rows affected (0.03 sec)
Records: 4  Duplicates: 0  Warnings: 0
SELECT*FROM DEPARTMENTS;
+---------------+-----------------+
| department_id | department_name |
+---------------+-----------------+
|             1 | Sales           |
```

```
|           2 | Marketing      |
|           3 | IT             |
|           4 | Finance        |
+-------------+----------------+
4 rows in set (0.00 sec)
INSERT INTO employees (employee_id, employee_name, department_id, salary)
VALUES (1, 'John Doe', 1, 50000), (2, 'Jane Doe', 2, 55000),
      (3, 'Jim Smith', 3, 60000), (4, 'Sarah Johnson', 4, 65000),
      (5, 'Michael Johnson', 1, 55000);
Query OK, 5 rows affected (0.03 sec)
Records: 5  Duplicates: 0  Warnings: 0
-- Select all employees from the employees table
SELECT * FROM employees;
+-------------+----------------+--------------+----------+
| employee_id | employee_name  | department_id | salary   |
+-------------+----------------+--------------+----------+
|           1 | John Doe       |            1 | 50000.00 |
|           2 | Jane Doe       |            2 | 55000.00 |
|           3 | Jim Smith      |            3 | 60000.00 |
|           4 | Sarah Johnson  |            4 | 65000.00 |
|           5 | Michael Johnson |           1 | 55000.00 |
+-------------+----------------+--------------+----------+
5 rows in set (0.00 sec)

-- Select all employees who make more than 55000
SELECT * FROM employees
WHERE salary > 55000;
+-------------+--------------+--------------+----------+
```

```
| employee_id | employee_name | department_id | salary   |
+-------------+---------------+---------------+----------+
|           3 | Jim Smith     |             3 | 60000.00 |
|           4 | Sarah Johnson |             4 | 65000.00 |
+-------------+---------------+---------------+----------+
2 rows in set (0.00 sec)
-- Select employee names and department names from both tables using a join
SELECT employees.employee_name, departments.department_name
FROM employees
JOIN departments
ON employees.department_id = departments.department_id;
+-----------------+-----------------+
| employee_name   | department_name |
+-----------------+-----------------+
| John Doe        | Sales           |
| Michael Johnson | Sales           |
| Jane Doe        | Marketing       |
| Jim Smith       | IT              |
| Sarah Johnson   | Finance         |
+-----------------+-----------------+
5 rows in set (0.00 sec)
-- Subquery to find the average salary of all employees
SELECT AVG(salary)
FROM (SELECT salary FROM employees) AS emp_salary;
+--------------+
| AVG(salary)  |
+--------------+
| 57000.000000 |
```

```
+--------------+
1 row in set (0.06 sec)
-- Simple join operation to combine data from both tables
SELECT employees.employee_name, departments.department_name, employees.salary
FROM employees
JOIN departments
ON employees.department_id = departments.department_id;
+-----------------+-----------------+----------+
| employee_name   | department_name | salary   |
+-----------------+-----------------+----------+
| John Doe        | Sales           | 50000.00 |
| Michael Johnson | Sales           | 55000.00 |
| Jane Doe        | Marketing       | 55000.00 |
| Jim Smith       | IT              | 60000.00 |
| Sarah Johnson   | Finance         | 65000.00 |
+-----------------+-----------------+----------+
5 rows in set (0.00 sec)
.
```

EX NO:5 QUERY DATABASE TABLES USING NATURAL, EQUI AND OUTER JOINS

```
CREATE TABLE customers (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  email VARCHAR(50)
);
Query OK, 0 rows affected (0.11 sec)
CREATE TABLE orders (
  id INT PRIMARY KEY,
  customer_id INT,
  product VARCHAR(50),
  price DECIMAL(10, 2),
  CONSTRAINT fk_customer
    FOREIGN KEY (customer_id)
    REFERENCES customers(id)
);
Query OK, 0 rows affected (0.16 sec)
INSERT INTO customers (id, name, email)
VALUES (1, 'Prakash', 'john.doe@example.com'),
       (2, 'Jane Smith', 'jane.smith@example.com'),
       (3, 'Bob Johnson', 'bob.johnson@example.com');
Query OK, 3 rows affected (0.08 sec)
Records: 3  Duplicates: 0  Warnings: 0
SELECT*FROM CUSTOMERS;
```

```
+----+-------------+------------------------+
| id | name        | email                  |
+----+-------------+------------------------+
|  1 | Prakash     | john.doe@example.com   |
|  2 | Jane Smith  | jane.smith@example.com |
|  3 | Bob Johnson | bob.johnson@example.com |
+----+-------------+------------------------+
3 rows in set (0.00 s
INSERT INTO orders (id, customer_id, product, price)
VALUES (1, 1, 'Widget', 10.00),
       (2, 2, 'Gizmo', 15.00),
       (3, 1, 'Thingamajig', 20.00),
       (4, 3, 'Widget', 10.00);
Query OK, 4 rows affected (0.06 sec)
Records: 4  Duplicates: 0  Warnings: 0
SELECT*FROM ORDERS;
+----+-------------+-------------+-------+
| id | customer_id | product     | price |
+----+-------------+-------------+-------+
|  1 |           1 | Widget      | 10.00 |
|  2 |           2 | Gizmo       | 15.00 |
|  3 |           1 | Thingamajig | 20.00 |
|  4 |           3 | Widget      | 10.00 |
+----+-------------+-------------+-------+
4 rows in set (0.00 sec)
SELECT *
FROM customers
NATURAL JOIN orders;
```

```
+----+-------------+-----------------------+-------------+-------------+-------+
| id | name        | email                 | customer_id | product     | price |
+----+-------------+-----------------------+-------------+-------------+-------+
|  1 | Prakash     | john.doe@example.com  |           1 | Widget      | 10.00 |
|  2 | Jane Smith  | jane.smith@example.com |           2 | Gizmo       | 15.00 |
|  3 | Bob Johnson | bob.johnson@example.com |           1 | Thingamajig | 20.00 |
+----+-------------+-----------------------+-------------+-------------+-------+
3 rows in set (0.05 sec)
SELECT *
FROM customers
JOIN orders
ON customers.id = orders.customer_id;
+----+-------------+-----------------------+----+-------------+-------------+-------+
| id | name        | email                 | id | customer_id | product     | price |
+----+-------------+-----------------------+----+-------------+-------------+-------+
|  1 | Prakash     | john.doe@example.com  |  1 |           1 | Widget      | 10.00 |
|  2 | Jane Smith  | jane.smith@example.com |  2 |           2 | Gizmo       | 15.00 |
|  1 | Prakash     | john.doe@example.com  |  3 |           1 | Thingamajig | 20.00 |
|  3 | Bob Johnson | bob.johnson@example.com |  4 |           3 | Widget      | 10.00 |
+----+-------------+-----------------------+----+-------------+-------------+-------+
4 rows in set (0.00 sec)
SELECT *
FROM orders
RIGHT OUTER JOIN customers ON orders.customer_id = customers.id;
+------+-------------+-------------+-------+----+-------------+-----------------------+
| id   | customer_id | product     | price | id | name        | email                 |
+------+-------------+-------------+-------+----+-------------+-----------------------+
|    1 |           1 | Widget      | 10.00 |  1 | Prakash     | john.doe@example.com  |
```

```
|    3 |          1 | Thingamajig | 20.00 |  1 | Prakash     | john.doe@example.com    |
|    2 |          2 | Gizmo       | 15.00 |  2 | Jane Smith  | jane.smith@example.com  |
|    4 |          3 | Widget      | 10.00 |  3 | Bob Johnson | bob.johnson@example.com |
+------+------------+-------------+-------+----+-------------+-------------------------+
4 rows in set (0.05 sec)

SELECT *
FROM customers
LEFT OUTER JOIN orders ON customers.id = orders.customer_id;
+----+-------------+-------------------------+------+-------------+-------------+-------+
| id | name        | email                   | id   | customer_id | product     | price |
+----+-------------+-------------------------+------+-------------+-------------+-------+
|  1 | Prakash     | john.doe@example.com    |    1 |           1 | Widget      | 10.00 |
|  1 | Prakash     | john.doe@example.com    |    3 |           1 | Thingamajig | 20.00 |
|  2 | Jane Smith  | jane.smith@example.com  |    2 |           2 | Gizmo       | 15.00 |
|  3 | Bob Johnson | bob.johnson@example.com |    4 |           3 | Widget      | 10.00 |
+----+-------------+-------------------------+------+-------------+-------------+-------+
4 rows in set (0.00 sec)
```

EX NO:6 WRITE USER DEFINED FUNCTIONS AND STORED PROCEDURES IN SQL.

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    hire_date DATE NOT NULL,
    salary DECIMAL(10,2) NOT NULL
);
Query OK, 0 rows affected (0.09 sec)

INSERT INTO employees (employee_id, first_name, last_name, hire_date, salary)
VALUES (1, 'ROSE', 'PARK', '2020-01-01', 50000.00),
 (2, 'LISA', 'MANABONAN', '2020-01-01', 50000.00),
 (4, 'JISOO', 'KIM', '2020-01-01', 50000.00),
(3, 'JENNY', 'KIM', '2019-01-01', 50000.00);
Query OK, 4 rows affected (0.08 sec)
Records: 4  Duplicates: 0  Warnings: 0
SELECT*FROM EMPLOYEES;
+-------------+------------+-----------+------------+----------+
| employee_id | first_name | last_name | hire_date  | salary   |
+-------------+------------+-----------+------------+----------+
|           1 | ROSE       | PARK      | 2020-01-01 | 50000.00 |
|           2 | LISA       | MANABONAN | 2020-01-01 | 50000.00 |
|           3 | JENNY      | KIM       | 2019-01-01 | 50000.00 |
|           4 | JISOO      | KIM       | 2020-01-01 | 50000.00 |
```

```
+-------------+------------+-----------+------------+----------+
4 rows in set (0.00 sec)
DELIMITER $$

CREATE FUNCTION getFullName(firstName VARCHAR(50), lastName VARCHAR(50))
RETURNS VARCHAR(100)
BEGIN
    DECLARE fullName VARCHAR(100);
    SET fullName = CONCAT(firstName, ' ', lastName);
    RETURN fullName;
END$$
Query OK, 0 rows affected (0.22 sec)
DELIMITER ;
SELECT*FROM EMPLOYEES;
DELIMITER //

CREATE PROCEDURE updateEmployeeSalary(IN employeeId INT, IN newSalary
DECIMAL(10,2))
BEGIN
    UPDATE employees
    SET salary = newSalary
    WHERE employee_id = employeeId;
END //
Query OK, 0 rows affected (0.05 sec)
DELIMITER ;
SELECT*FROM EMPLOYEES;
+-------------+------------+-----------+------------+----------+
| employee_id | first_name | last_name | hire_date  | salary   |
+-------------+------------+-----------+------------+----------+
```

| 1 | ROSE | PARK | 2020-01-01 | 50000.00 |

| 2 | LISA | MANABONAN | 2020-01-01 | 50000.00 |

| 3 | JENNY | KIM | 2019-01-01 | 50000.00 |

| 4 | JISOO | KIM | 2020-01-01 | 50000.00 |

+-------------+------------+-----------+------------+----------+

4 rows in set (0.00 s)

CALL updateEmployeeSalary(1, 55000.00);

Query OK, 1 row affected (0.09 sec)

SELECT*FROM EMPLOYEES;

employee_id | first_name | last_name | hire_date | salary |

+-------------+------------+-----------+------------+----------+

| 1 | ROSE | PARK | 2020-01-01 | 55000.00 |

| 2 | LISA | MANABONAN | 2020-01-01 | 50000.00 |

| 3 | JENNY | KIM | 2019-01-01 | 50000.00 |

| 4 | JISOO | KIM | 2020-01-01 | 50000.00 |

+-------------+------------+-----------+------------+----------+

4 rows in set (0.00 sec)

EX NO:7 EXECUTE COMPLEX TRANSACTIONS AND REALIZE DCL AND TCL COMMANDS.

```
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    hire_date DATE NOT NULL,
    salary DECIMAL(10,2) NOT NULL
);
```

Query OK, 0 rows affected (0.13 sec)

```
START TRANSACTION;
```

Query OK, 0 rows affected (0.00 sec)

```
INSERT INTO employees (employee_id, first_name, last_name, hire_date, salary)
VALUES (1, 'ROSE', 'PARK', '2020-01-01', 50000.00), (2, 'LISA', 'MANABONAN', '2020-01-01', 50000.00),(3, 'JENNY', 'KIM', '2019-01-01', 50000.00), (4, 'JISOO', 'KIM', '2019-01-01', 50000.00);
```

Query OK, 4 rows affected (0.00 sec)

Records: 4 Duplicates: 0 Warnings: 0

```
SELECT*FROM EMPLOYEES;
```

```
+-------------+------------+-----------+------------+----------+
| employee_id | first_name | last_name | hire_date  | salary   |
+-------------+------------+-----------+------------+----------+
|           1 | ROSE       | PARK      | 2020-01-01 | 50000.00 |
|           2 | LISA       | MANABONAN | 2020-01-01 | 50000.00 |
|           3 | JENNY      | KIM       | 2019-01-01 | 50000.00 |
|           4 | JISOO      | KIM       | 2019-01-01 | 50000.00 |
+-------------+------------+-----------+------------+----------+
```

4 rows in set (0.00 sec)

```
UPDATE employees
SET salary = salary * 1.1
WHERE hire_date < '2021-01-01';
Query OK, 4 rows affected (0.05 sec)
Rows matched: 4  Changed: 4  Warnings: 0
SELECT*FROM EMPLOYEES;
+-------------+------------+-----------+------------+----------+
| employee_id | first_name | last_name | hire_date  | salary   |
+-------------+------------+-----------+------------+----------+
|           1 | ROSE       | PARK      | 2020-01-01 | 55000.00 |
|           2 | LISA       | MANABONAN | 2020-01-01 | 55000.00 |
|           3 | JENNY      | KIM       | 2019-01-01 | 55000.00 |
|           4 | JISOO      | KIM       | 2019-01-01 | 55000.00 |
+-------------+------------+-----------+------------+----------+
4 rows in set (0.00 sec)
COMMIT;
Query OK, 0 rows affected (0.06 sec)
```

EX NO:8 WRITE SQL TRIGGERS FOR INSERT, DELETE, AND UPDATE OPERATIONS IN A DATABASE TABLE.

```
-- Create a table
CREATE TABLE employees (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  department VARCHAR(50),
  salary INT
);
Query OK, 0 rows affected (0.06 sec)
CREATE TABLE employee_audit (
  id INT PRIMARY KEY AUTO_INCREMENT,
  action VARCHAR(10),
  action_date TIMESTAMP
);
 Query OK, 0 rows affected (0.16 sec)
-- Change delimiter to //
DELIMITER //

-- Create a trigger for insert operations
CREATE TRIGGER insert_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
```

```
  INSERT INTO employee_audit (id, action, action_date)
  VALUES (NEW.id, 'insert', NOW());
END//
Query OK, 0 rows affected (0.14 sec)

-- Create a trigger for delete operations
CREATE TRIGGER delete_employee
AFTER DELETE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_audit (id, action, action_date)
  VALUES (OLD.id, 'delete', NOW());
END//
Query OK, 0 rows affected (0.09 sec)
-- Create a trigger for update operations
CREATE TRIGGER update_employee
AFTER UPDATE ON employees
FOR EACH ROW
BEGIN
  INSERT INTO employee_audit (id, action, action_date)
  VALUES (NEW.id, 'update', NOW());
END//
Query OK, 0 rows affected (0.08 sec)
-- Restore delimiter to ;
DELIMITER ;
INSERT INTO employees (id, name, department, salary) VALUES
```

(1, 'John Smith', 'Sales', 50000),

(2, 'Jane Doe', 'Marketing', 60000),

(3, 'Bob Johnson', 'HR', 55000),

(4, 'Sarah Lee', 'IT', 65000),

(5, 'PRAKASH SN', 'Finance', 70000);

SELECT*FROM EMPLOYEES;

```
+----+-------------+------------+--------+
| id | name        | department | salary |
+----+-------------+------------+--------+
|  1 | John Smith  | Sales      |  50000 |
|  2 | Jane Doe    | Marketing  |  60000 |
|  3 | Bob Johnson | HR         |  55000 |
|  4 | Sarah Lee   | IT         |  65000 |
|  5 | PRAKASH SN  | Finance    |  70000 |
+----+-------------+------------+--------+
```

5 rows in set (0.00 sec)

SELECT*FROM EMPLOYEE_AUDIT;

```
+----+--------+---------------------+
| id | action | action_date         |
+----+--------+---------------------+
|  1 | insert | 2023-04-27 17:12:04 |
|  2 | insert | 2023-04-27 17:12:04 |
|  3 | insert | 2023-04-27 17:12:04 |
|  4 | insert | 2023-04-27 17:12:04 |
|  5 | insert | 2023-04-27 17:12:04 |
```

```
+----+--------+--------------------+
```

5 rows in set (0.00 sec)

EX NO: 9 Create View and index for database tables with a large number of records

```
-- create the table
CREATE TABLE users (
  id INT PRIMARY KEY,
  name VARCHAR(50),
  email VARCHAR(100)
);
Query OK, 0 rows affected (0.16 sec)
-- insert 10 sample values
INSERT INTO users (id, name, email)
VALUES
  (1, 'John Smith', 'john@example.com'),
  (2, 'Jane Doe', 'jane@example.com'),
  (3, 'Bob Johnson', 'bob@example.com'),
  (4, 'Mary Williams', 'mary@example.com'),
  (5, 'David Brown', 'david@example.com'),
  (6, 'Karen Davis', 'karen@example.com'),
  (7, 'Tom Wilson', 'tom@example.com'),
  (8, 'Lisa Lee', 'lisa@example.com'),
  (9, 'Mike Chen', 'mike@example.com'),
  (10, 'Amy Taylor', 'amy@example.com');
Query OK, 10 rows affected (0.03 sec)
Records: 10  Duplicates: 0  Warnings: 0
-- create the view
CREATE VIEW user_emails AS
```

```
SELECT name, email
FROM users;
Query OK, 0 rows affected (0.03 sec)
-- create an index on the email column
CREATE INDEX email_index ON users (email);
Query OK, 0 rows affected (0.16 sec)
Records: 0  Duplicates: 0  Warnings: 0
-- query the view
SELECT * FROM user_emails;
+---------------+-------------------+
| name          | email             |
+---------------+-------------------+
| John Smith    | john@example.com  |
| Jane Doe      | jane@example.com  |
| Bob Johnson   | bob@example.com   |
| Mary Williams | mary@example.com  |
| David Brown   | david@example.com |
| Karen Davis   | karen@example.com |
| Tom Wilson    | tom@example.com   |
| Lisa Lee      | lisa@example.com  |
| Mike Chen     | mike@example.com  |
| Amy Taylor    | amy@example.com   |
+---------------+-------------------+
10 rows in set (0.03 sec)
-- query the table using the index
SELECT * FROM users WHERE email = 'john@example.com';
+----+------------+------------------+
| id | name       | email            |
```

```
+----+------------+------------------+
|  1 | John Smith | john@example.com |
+----+------------+------------------+
1 row in set (0.00 sec)
```

EX NO:10 CREATE AN XML DATABASE AND VALIDATE IT USING XML SCHEMA.

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book>
    <title>Harry Potter and the Philosopher's Stone</title>
    <author>J.K. Rowling</author>
    <year>1997</year>
    <publisher>Bloomsbury</publisher>
  </book>
  <book>
    <title>The Lord of the Rings</title>
    <author>J.R.R. Tolkien</author>
    <year>1954</year>
    <publisher>Allen &amp;amp; Unwin</publisher>
  </book>
</books>
```

xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="books">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string"/>
              <xs:element name="author" type="xs:string"/>
              <xs:element name="year" type="xs:integer"/>
```

```
            <xs:element name="publisher" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

```
import xmlschema

# Load the schema file
schema = xmlschema.XMLSchema('books.xsd')

# Load the XML file
xml = 'books.xml'

# Validate the XML file against the schema
if schema.is_valid(xml):
    print('Validation successful')
else:
    print('Validation failed')
```

EX NO: 11 CREATE DOCUMENT, COLUMN AND GRAPH BASED DATA USING NOSQL DATABASE TOOLS

AIM:

To create document, column and graph based data using NOSQL database tools

ALGORITHM:

1. Choose a NoSQL database tool that supports the type of data model you need (document, column, or graph). Examples of popular NoSQL database tools include MongoDB, Cassandra, and Neo4j.
2. Install and configure the NoSQL database tool according to the instructions provided by the vendor. This typically involves setting up a database instance, defining security settings, and configuring the database for optimal performance.
3. For document-based data, create a database and collection within the NoSQL database tool. Define the structure of your documents using a schema or by allowing the database to create a flexible schema on-the-fly. Insert documents into the collection using the tool's API or a client application.
4. For column-based data, create a keyspace and column family within the NoSQL database tool. Define the structure of your columns using a schema or by allowing the database to create a flexible schema on-the-fly. Insert columns into the column family using the tool's API or a client application.
5. For graph-based data, create a graph within the NoSQL database tool. Define the structure of your nodes and edges using a schema or by allowing the database to create a flexible schema on-the-fly. Insert nodes and edges into the graph using the tool's API or a client application.
6. Create indexes and views to optimize queries against your data. NoSQL databases typically provide flexible indexing options that allow you to optimize queries based on the type of data you are working with.
7. Use the tool's API or client applications to retrieve and manipulate data stored in the NoSQL database. The API will vary depending on the specific NoSQL database tool you are using.
8. Periodically backup your NoSQL database to ensure data is protected in case of system failure or other issues.
9. Monitor performance of the NoSQL database tool to ensure it is performing optimally. NoSQL databases can be highly scalable and can handle large amounts of data, but it is important to monitor performance to ensure the database remains responsive as data volumes grow.

RESULT: Thus the create document, column and graph based data using NOSQL database tools has been created and executed successfully.

EX NO: 12 DEVELOP A SIMPLE GUI BASED DATABASE APPLICATION AND INCORPORATE ALL THE ABOVE-MENTIONED FEATURES

AIM:

To develop a simple GUI based database application and incorporate all the above-mentioned features.

ALGORITHM:

1. Choose a programming language and GUI framework to develop your application. Popular choices include Java with Swing or JavaFX, Python with PyQt or Tkinter, and C# with Windows Forms or WPF.
2. Choose a NoSQL database tool that supports the type of data model you need (document, column, or graph). Examples of popular NoSQL database tools include MongoDB for document-based data, Apache Cassandra for column-based data, and Neo4j for graph-based data.
3. Install and configure the NoSQL database tool according to the instructions provided by the vendor. This typically involves setting up a database instance, defining security settings, and configuring the database for optimal performance.
4. Design the GUI for your application using the GUI framework of your choice. Your GUI should allow users to view, add, update, and delete data stored in the NoSQL database.
5. Implement code to connect to the NoSQL database from your application. This code will vary depending on the specific NoSQL database tool you are using.
6. Implement code to retrieve and display data from the NoSQL database in the GUI. This code will also vary depending on the specific NoSQL database tool you are using and the type of data model (document, column, or graph-based) you are working with.
7. Implement code to add, update, and delete data in the NoSQL database from the GUI. This code will also vary depending on the specific NoSQL database tool you are using and the type of data model you are working with.
8. Implement indexing and view options in your application to optimize queries against the data stored in the NoSQL database. This will involve using the indexing features provided by your NoSQL database tool.
9. Test your application to ensure it is functioning properly and data is being stored and retrieved correctly from the NoSQL database.

By following these steps, you can develop a simple GUI-based database application that incorporates document, column, and graph-based data models. The specific steps and tools used will depend on the requirements of your project and the specific NoSQL database tool being used.

RESULT:

Thus to Develop a simple GUI based database application and incorporate all the above-mentioned features has been written and executed successfully.