```
--To create a database and signup and login tables with all the field requirements, and then
implement insert, update, select,
--and delete operations by using SQL queries.


CREATE DATABASE Claysys;
USE Claysys;



CREATE TABLE signup (
   userid INT IDENTITY(1,1) PRIMARY KEY,
   firstname VARCHAR(50) NOT NULL,
   lastname VARCHAR(50) NOT NULL,
   dateofbirth DATE NOT NULL,
   age INT NOT NULL,
   gender VARCHAR(10) NOT NULL CHECK (gender IN ('Male', 'Female', 'Other')),
   phonenumber VARCHAR(15) UNIQUE NOT NULL,
   emailaddress VARCHAR(50) UNIQUE NOT NULL,
   useraddress TEXT NOT NULL,
   userstate VARCHAR(50) NOT NULL,
   city VARCHAR(50) NOT NULL,
   username VARCHAR(50) UNIQUE NOT NULL,
   originalpassword VARCHAR(50) NOT NULL,
   confirmpassword VARCHAR(50) NOT NULL );




CREATE TABLE login (
   login_id INT IDENTITY(1,1) PRIMARY KEY,
   username VARCHAR(50) UNIQUE NOT NULL,
   originalpassword VARCHAR(255) NOT NULL,
   email_address VARCHAR(100) UNIQUE NOT NULL);


INSERT INTO signup (firstname, lastname, dateofbirth, age, gender, phonenumber,
emailaddress, useraddress, userstate, city, username, originalpassword, confirmpassword)
VALUES ('Haripriya', 'Nair', '1998-07-25', 26, 'Female', '9876543210', 'haripriya@example.com',
'123 Street, Area', 'Kerala', 'Kochi', 'haripriya_rn', 'securepassword', 'securepassword');
```

```
INSERT INTO signup (firstname, lastname, dateofbirth, age, gender, phonenumber,
emailaddress, useraddress, userstate, city, username, originalpassword, confirmpassword)
VALUES('Rahul' , 'Raj', '1990-04-25', 30, 'Male', '3667788720', 'rohan@example.com', ' Street,
Area', 'Kerala', 'Kochi', 'rahul@', 'password', 'password'),
('Mani', 'M', '2002-03-25', 22, 'Male', '3666693460', 'mani@example.com', ' Street3, Area',
'Kerala', 'Kochi', 'roh_n', 'password123', 'password123');
```

```
INSERT INTO login (username, originalpassword, email_address)
VALUES ('haripriya_rn', 'securepassword', 'haripriya@example.com'),( 'rahul@',
'password','rohan@example.com'),('roh_n', 'password123','mani@example.com');
```

```
UPDATE signup
SET emailaddress = 'newemail@example.com', phonenumber = '9876543211'
WHERE userid = 1; -- Replace 1 with the appropriate user_id
```

```
SELECT * FROM login
WHERE username = 'haripriya_rn';
```

```
DELETE FROM signup
WHERE userid = 1; -- Replace 1 with the appropriate user_id
```

```
select * from signup;
select * from login;
------------------------------------------------------------------------------------------------------------------------
----------------------
```

```
--To create an employee table and how to achieve or get the second highest salary from the
table.
```

```
CREATE TABLE employee (
    emp_id INT IDENTITY(1,1) PRIMARY KEY,
    emp_name VARCHAR(100) NOT NULL,
    salary DECIMAL(10,2) NOT NULL);


INSERT INTO employee (emp_name, salary)
VALUES
('John Doe', 5000),
('Jane Smith', 6000),
('Alice Brown', 7000),
('Bob Johnson', 8000),
('Charlie White', 9000);

SELECT DISTINCT salary
FROM employee
ORDER BY salary DESC
OFFSET 1 ROW FETCH NEXT 1 ROW ONLY;


SELECT MAX(salary) AS second_highest_salary
FROM employee
WHERE salary < (SELECT MAX(salary) FROM employee);


SELECT * FROM employee;
----------------------------------------------------------------------------------------------------------------------
-----------------------------



--Perform the SQL query to list the number of employees in each department.

ALTER TABLE employee
ADD department VARCHAR(50) ;



INSERT INTO employee (emp_name, salary, department)
VALUES ('John Doe', 5500, 'Marketing'),
('John Doe', 5500, 'Sales'),
('Jane Smith', 5500, 'HR'),
('Alice Brown', 5500, 'Software'),
('Bob Johnson', 5500, 'Marketing');


SELECT * FROM employee;
---------------------------------------------------------------------------------------------------------------------------
--------------------------



--To create two tables and implement all the SQL join concepts.
CREATE TABLE employees (
   emp_id INT IDENTITY(1,1) PRIMARY KEY,
   emp_name VARCHAR(100) NOT NULL,
   salary DECIMAL(10,2) NOT NULL,
   department_id INT );


CREATE TABLE departments (
   department_id INT PRIMARY KEY,
   department_name VARCHAR(100) NOT NULL);

INSERT INTO departments (department_id, department_name)
VALUES
(1, 'HR'),
(2, 'Finance'),
(3, 'IT'),
(4, 'Marketing');


INSERT INTO employees (emp_name, salary, department_id)
VALUES
('John Doe', 5000, 1),
('Jane Smith', 6000, 2),
('Alice Brown', 7000, 3),
('Bob Johnson', 8000, 3),
('Charlie White', 9000, NULL); -- Employee without a department


SELECT *FROM employees;
SELECT * FROM departments;
SELECT e.emp_name, e.salary, d.department_name
FROM employees e
INNER JOIN departments d
```

```
ON e.department_id = d.department_id;


SELECT e.emp_name, e.salary, d.department_name
FROM employees e
LEFT JOIN departments d
ON e.department_id = d.department_id;

SELECT e.emp_name, e.salary, d.department_name
FROM employees e
RIGHT JOIN departments d
ON e.department_id = d.department_id;

SELECT e.emp_name, e.salary, d.department_name
FROM employees e
FULL OUTER JOIN departments d
ON e.department_id = d.department_id;

SELECT e.emp_name, d.department_name
FROM employees e
CROSS JOIN departments d;


SELECT e1.emp_name AS employee1, e2.emp_name AS employee2, d.department_name
FROM employees e1
INNER JOIN employees e2
ON e1.department_id = e2.department_id AND e1.emp_id != e2.emp_id
INNER JOIN departments d
ON e1.department_id = d.department_id;-- Pairs of employees in the same department.(self
join)
------------------------------------------------------------------------------------------------------------------------------------------------


--6. To create different stored procedures for implementing the CRUD operations on the sign-up
page.

----------------------C
CREATE PROCEDURE InsertSignup
    @Firstname VARCHAR(50),
    @Lastname VARCHAR(50),
    @DateOfBirth DATE,
    @Age INT,
    @Gender VARCHAR(10),
    @PhoneNumber VARCHAR(15),
    @EmailAddress VARCHAR(50),
    @UserAddress TEXT,
    @UserState VARCHAR(50),
    @City VARCHAR(50),
    @Username VARCHAR(50),
    @OriginalPassword VARCHAR(50),
    @ConfirmPassword VARCHAR(50)
AS
BEGIN
    INSERT INTO signup (firstname, lastname, dateofbirth, age, gender, phonenumber,
emailaddress, useraddress, userstate, city, username, originalpassword, confirmpassword)
    VALUES (@Firstname, @Lastname, @DateOfBirth, @Age, @Gender, @PhoneNumber,
@EmailAddress, @UserAddress, @UserState, @City, @Username, @OriginalPassword,
@ConfirmPassword);
END;

EXEC InsertSignup
    'Haripriya',
    'Nair',
    '1998-07-25',
    26,
    'Female',
    '9876543210',
    'haripriya@example.com',
    '123 Street',
    'Kerala',
    'Kochi',
    'haripriya_rn',
    'securepassword',
    'securepassword';


          ------------R
          CREATE PROCEDURE GetAllSignupData
AS
BEGIN
    SELECT * FROM signup;
END;


CREATE PROCEDURE GetSignupDataByID
    @UserID INT
AS
```

```
BEGIN
   SELECT * FROM signup WHERE userid = @UserID;
END;


------------------------------------U
CREATE PROCEDURE UpdateSignup
   @UserID INT,
   @Firstname VARCHAR(50),
   @Lastname VARCHAR(50),
   @PhoneNumber VARCHAR(15),
   @EmailAddress VARCHAR(50),
   @UserAddress TEXT,
   @UserState VARCHAR(50),
   @City VARCHAR(50)
AS
BEGIN
   UPDATE signup
   SET firstname = @Firstname,
      lastname = @Lastname,
      phonenumber = @PhoneNumber,
      emailaddress = @EmailAddress,
      useraddress = @UserAddress,
      userstate = @UserState,
      city = @City
   WHERE userid = @UserID;
END;


EXEC UpdateSignup
   @UserID = 1,
   @Firstname = 'Haripriya',
   @Lastname = 'Nair',
   @PhoneNumber = '9876543211',
   @EmailAddress = 'haripriya_updated@example.com',
   @UserAddress = '456 New Street',
   @UserState = 'Kerala',
   @City = 'Ernakulam';


         -------------------------------D
         CREATE PROCEDURE DeleteSignup
   @UserID INT
AS
```

```
BEGIN
   DELETE FROM signup WHERE userid = @UserID;
END;

EXEC DeleteSignup @UserID = 1;
```

-- To create a single stored procedure for implementing the CRUD operations on the Student Admission Form page.

```
CREATE TABLE StudentAdmission (
   student_id INT IDENTITY(1,1) PRIMARY KEY,
   first_name VARCHAR(50) NOT NULL,
   last_name VARCHAR(50) NOT NULL,
   date_of_birth DATE NOT NULL,
   age INT NOT NULL,
   gender VARCHAR(10) NOT NULL CHECK (gender IN ('Male', 'Female', 'Other')),
   phone_number VARCHAR(15) UNIQUE NOT NULL,
   email_address VARCHAR(50) UNIQUE NOT NULL,
   address TEXT NOT NULL,
   state VARCHAR(50) NOT NULL,
   city VARCHAR(50) NOT NULL);

         CREATE PROCEDURE ManageStudentAdmission
   @Operation NVARCHAR(10), -- Type of operation: 'CREATE', 'READ', 'UPDATE', 'DELETE'
   @StudentID INT = NULL, -- Needed for READ, UPDATE, DELETE
   @FirstName VARCHAR(50) = NULL, -- Needed for CREATE and UPDATE
   @LastName VARCHAR(50) = NULL,
   @DateOfBirth DATE = NULL,
   @Age INT = NULL,
   @Gender VARCHAR(10) = NULL,
   @PhoneNumber VARCHAR(15) = NULL,
   @EmailAddress VARCHAR(50) = NULL,
   @Address TEXT = NULL,
   @State VARCHAR(50) = NULL,
   @City VARCHAR(50) = NULL
AS
BEGIN
   -- CREATE Operation
   IF @Operation = 'CREATE'
   BEGIN
```

```
        INSERT INTO StudentAdmission (first_name, last_name, date_of_birth, age, gender,
phone_number, email_address, address, state, city)
        VALUES (@FirstName, @LastName, @DateOfBirth, @Age, @Gender, @PhoneNumber,
@EmailAddress, @Address, @State, @City);
    END

    -- READ Operation
    ELSE IF @Operation = 'READ'
    BEGIN
        IF @StudentID IS NULL
        BEGIN
            -- Retrieve all records
            SELECT * FROM StudentAdmission;
        END
        ELSE
        BEGIN
            -- Retrieve specific record by StudentID
            SELECT * FROM StudentAdmission WHERE student_id = @StudentID;
        END
    END

    -- UPDATE Operation
    ELSE IF @Operation = 'UPDATE'
    BEGIN
        UPDATE StudentAdmission
        SET first_name = @FirstName,
            last_name = @LastName,
            date_of_birth = @DateOfBirth,
            age = @Age,
            gender = @Gender,
            phone_number = @PhoneNumber,
            email_address = @EmailAddress,
            address = @Address,
            state = @State,
            city = @City
        WHERE student_id = @StudentID;
    END

    -- DELETE Operation
    ELSE IF @Operation = 'DELETE'
    BEGIN
        DELETE FROM StudentAdmission WHERE student_id = @StudentID;
    END
END;
```

```
EXEC ManageStudentAdmission
   @Operation = 'CREATE',
   @FirstName = 'Haripriya',
   @LastName = 'Nair',
   @DateOfBirth = '1998-07-25',
   @Age = 26,
   @Gender = 'Female',
   @PhoneNumber = '9876543210',
   @EmailAddress = 'haripriya@example.com',
   @Address = '123 Street, Kochi',
   @State = 'Kerala',
   @City = 'Kochi';

        -- Retrieve all records
EXEC ManageStudentAdmission @Operation = 'READ';

-- Retrieve a specific record
EXEC ManageStudentAdmission @Operation = 'READ', @StudentID = 1;


EXEC ManageStudentAdmission
   @Operation = 'UPDATE',
   @StudentID = 1,
   @FirstName = 'Haripriya',
   @LastName = 'R Nair',
   @DateOfBirth = '1998-07-25',
   @Age = 27,
   @Gender = 'Female',
   @PhoneNumber = '9876543211',
   @EmailAddress = 'updated_haripriya@example.com',
   @Address = '456 New Street, Ernakulam',
   @State = 'Kerala',
   @City = 'Ernakulam';

        EXEC ManageStudentAdmission @Operation = 'DELETE', @StudentID = 1;
---------------------------------------------------------------------------------------------------------------------------
-----
```

--8. To Learn and implementing the example queries or the following tasks:

```
--a. Normalization Techniques - All the normal forms

--1NF:
CREATE TABLE Students (
StudentID INT,
Name VARCHAR(50),
Subject VARCHAR(50));

INSERT INTO Students (StudentID, Name, Subject)
VALUES(1, 'Haripriya', 'Math'),
   (1, 'Haripriya', 'Science'),
   (2, 'John', 'English'),
   (2, 'John', 'Math');

SELECT * FROM Students;



--2NF
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   StudentName VARCHAR(50)
);

CREATE TABLE Courses (
   CourseID INT PRIMARY KEY,
   CourseName VARCHAR(50)
);

CREATE TABLE Enrollment (
   StudentID INT,
   CourseID INT,
   FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
   FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
);

-- Insert Data
INSERT INTO Students (StudentID, StudentName)
VALUES (1, 'Haripriya'), (2, 'John');

INSERT INTO Courses (CourseID, CourseName)
VALUES (101, 'Math'), (102, 'Science');

INSERT INTO Enrollment (StudentID, CourseID)
```

```sql
VALUES (1, 101), (1, 102);


--3NF
CREATE TABLE Instructors (
   InstructorName VARCHAR(50) PRIMARY KEY,
   Department VARCHAR(50)
);

ALTER TABLE Enrollment
ADD InstructorName VARCHAR(50),
FOREIGN KEY (InstructorName) REFERENCES Instructors(InstructorName);

-- Insert Data
INSERT INTO Instructors (InstructorName, Department)
VALUES ('Mr. Smith', 'Science'), ('Dr. Brown', 'Arts');

UPDATE Enrollment
SET InstructorName = 'Mr. Smith' WHERE CourseID = 101;

UPDATE Enrollment
SET InstructorName = 'Dr. Brown' WHERE CourseID = 102;

--4NF
CREATE TABLE CourseTime (
   CourseID INT PRIMARY KEY,
   TimeSlot VARCHAR(50)
);

CREATE TABLE CourseInstructor (
   CourseID INT PRIMARY KEY,
   InstructorName VARCHAR(50),
   FOREIGN KEY (InstructorName) REFERENCES Instructors(InstructorName)
);

-- Insert Data
INSERT INTO CourseTime (CourseID, TimeSlot)
VALUES (101, 'Morning'), (102, 'Evening');

INSERT INTO CourseInstructor (CourseID, InstructorName)
VALUES (101, 'Mr. Smith'), (102, 'Dr. Brown');
```

-

```
--b. Indexing Cluster and non-cluster indexing
-- Create a Products table
CREATE TABLE Products (
    ProductID INT PRIMARY KEY, -- Primary key automatically creates a clustered index
    ProductName VARCHAR(100) NOT NULL,
    Category VARCHAR(50) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL,
    StockQuantity INT NOT NULL);

-- Add a non-clustered index to the Category column
CREATE NONCLUSTERED INDEX IDX_Products_Category ON Products(Category);

-- Insert sample data into the Products table
INSERT INTO Products (ProductID, ProductName, Category, Price, StockQuantity)
VALUES
(101, 'Laptop', 'Electronics', 75000.00, 10),
(102, 'Smartphone', 'Electronics', 30000.00, 25),
(103, 'Refrigerator', 'Appliances', 50000.00, 15),
(104, 'Microwave Oven', 'Appliances', 10000.00, 20),
(105, 'T-shirt', 'Clothing', 500.00, 100),
(106, 'Jeans', 'Clothing', 1200.00, 50);



SELECT *
FROM Products
WHERE ProductID = 101; -- Uses the clustered index on ProductID

SELECT *
FROM Products
WHERE Category = 'Electronics'; -- Uses the non-clustered index on Category



--c. Pivot and unpivot the values in SQL Table

-- Create the Sales table
CREATE TABLE Sales (
SalesID INT PRIMARY KEY,
ProductName VARCHAR(50),
Month VARCHAR(20),
SalesAmount DECIMAL(10, 2));
```

```
-- Insert data into the Sales table
INSERT INTO Sales (SalesID, ProductName, Month, SalesAmount)
VALUES
(1, 'Laptop', 'January', 1000.00),
(2, 'Laptop', 'February', 1200.00),
(3, 'Laptop', 'March', 1100.00),
(4, 'Smartphone', 'January', 800.00),
(5, 'Smartphone', 'February', 950.00),
(6, 'Smartphone', 'March', 900.00);


-- Pivot Sales data to show Months as columns
SELECT ProductName,
[January] AS JanuarySales,
[February] AS FebruarySales,
[March] AS MarchSales
FROM (SELECT ProductName, Month, SalesAmount
FROM Sales) AS SourceTable
PIVOT (SUM(SalesAmount) FOR Month IN ([January], [February], [March])) AS PivotTable;


-- Unpivot Sales data to show Month and SalesAmount as rows
SELECT ProductName, Month, SalesAmount
FROM (SELECT ProductName, [January], [February], [March]
FROM (SELECT ProductName, Month, SalesAmount FROM Sales) AS SourceTable
PIVOT (SUM(SalesAmount) FOR Month IN ([January], [February], [March])) AS PivotTable) AS
PivotedTable
UNPIVOT (SalesAmount FOR Month IN ([January], [February], [March])) AS UnpivotedTable;


--d. Merge concepts in SQL Table

CREATE TABLE Products (
   ProductID INT PRIMARY KEY,
   ProductName VARCHAR(100) NOT NULL,
   Category VARCHAR(50) NOT NULL,
   Price DECIMAL(10, 2) NOT NULL,
   StockQuantity INT NOT NULL);

INSERT INTO Products (ProductID, ProductName, Category, Price, StockQuantity)
VALUES
(1, 'Laptop', 'Electronics', 75000.00, 10),
(2, 'Smartphone', 'Electronics', 30000.00, 25),
(3, 'Refrigerator', 'Appliances', 50000.00, 15);
```

```
CREATE TABLE ProductsUpdates (
   ProductID INT PRIMARY KEY,
   ProductName VARCHAR(100) NOT NULL,
   Category VARCHAR(50) NOT NULL,
   Price DECIMAL(10, 2) NOT NULL,
   StockQuantity INT NOT NULL);

INSERT INTO ProductsUpdates (ProductID, ProductName, Category, Price, StockQuantity)
VALUES
(2, 'Smartphone', 'Electronics', 28000.00, 30), -- Updated Price and StockQuantity
(3, 'Refrigerator', 'Appliances', 52000.00, 12), -- Updated Price and StockQuantity
(4, 'Microwave Oven', 'Appliances', 15000.00, 20); -- New product

MERGE INTO Products AS Target
USING ProductsUpdates AS Source
ON Target.ProductID = Source.ProductID

-- Update existing rows
WHEN MATCHED THEN
   UPDATE SET
      Target.ProductName = Source.ProductName,
      Target.Category = Source.Category,
      Target.Price = Source.Price,
      Target.StockQuantity = Source.StockQuantity

-- Insert new rows
WHEN NOT MATCHED BY TARGET THEN
   INSERT (ProductID, ProductName, Category, Price, StockQuantity)
   VALUES (Source.ProductID, Source.ProductName, Source.Category, Source.Price,
Source.StockQuantity)
```