**SQL basic programs for practice**

**1 ) create database**

--CREATE DATABASE SampleDB;

**2) Drop database**

--DROP DATABASE SampleDB;

**3 ) Create table**

CREATE TABLE Student (

StudentID INT PRIMARY KEY,

Name VARCHAR(50),

Age INT,

Grade VARCHAR(2),

TotalMarks INT,

City VARCHAR(50),

EnrollmentDate DATE

);

**4) Insert data**

INSERT INTO Student (StudentID, Name, Age, Grade, TotalMarks, City, EnrollmentDate)

VALUES (1, 'Sree', 19, 'B', 400 , 'kollam', '2024-08-02'),

(2, 'Amal', 19, 'c', 400 , 'konni', '2024-05-02'),

(3, 'vishnu', 18, 'B', 300 , 'trivandrum', '2024-08-23'),

(4, 'ajin', 19, 'A', 350 , 'wayanad', '2024-04-02'),

(5, 'vimal', 20, 'c', 100 , 'kollam', '2024-08-02');

**5) some random operations**

**Select all data**

-- SELECT \* FROM Student;

**Avoid duplicate city**

--SELECT DISTINCT City FROM Student;

**Student with age greater than 18**

--SELECT \* FROM Student WHERE Age > 18;

**Display data based on desending order of total mark**

--SELECT \* FROM Student ORDER BY TotalMarks DESC;

**Where condition**

--SELECT \* FROM Student WHERE (Age > 18 AND Grade = 'A') ;

**update**

--UPDATE Student SET TotalMarks = 480 WHERE StudentID = 1;

**Delete data**

--DELETE FROM Student WHERE StudentID = 3;

**Top in sql**

--SELECT TOP 3 \* FROM Student ORDER BY TotalMarks ;

**Aggregate functions**

-- SELECT MIN(TotalMarks) FROM Student;

--SELECT COUNT(\*) AS TotalStudents FROM Student;

--SELECT \* FROM Student WHERE Name LIKE 's%';

**Altertation in table**

--ALTER TABLE Student ADD Gender VARCHAR(20);

--ALTER TABLE Student DROP COLUMN Gender;

**View in sql**

CREATE VIEW [Age Greater Than 18]

AS

SELECT Age , Name

FROM Student

WHERE Age > 18;

SELECT \* FROM [Age Greater Than 18]

**Random practices questions**

USE Ecommerce;

CREATE TABLE Categories (

CategoryID INT IDENTITY(1,1) PRIMARY KEY,

CategoryName VARCHAR(50) NOT NULL

);

CREATE TABLE Products (

ProductID INT IDENTITY(1,1) PRIMARY KEY,

ProductName VARCHAR(50) NOT NULL,

Price DECIMAL(10, 2) NOT NULL,

Stock INT,

CategoryID INT FOREIGN KEY REFERENCES Categories(CategoryID)

);

CREATE TABLE Customers (

CustomerID INT IDENTITY(1,1) PRIMARY KEY,

CustomerName VARCHAR(50) NOT NULL,

City VARCHAR(50),

Email VARCHAR(100),

PhoneNumber VARCHAR(20)

);

CREATE TABLE Orders (

OrderID INT IDENTITY(1,1) PRIMARY KEY,

OrderDate DATE NOT NULL,

CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),

ProductID INT FOREIGN KEY REFERENCES Products(ProductID),

Quantity INT NOT NULL

);

**-- Inserting into Categories**

INSERT INTO Categories (CategoryName) VALUES

('Electronics'), ('Furniture'), ('Clothing'), ('Books');

**-- Inserting into Products**

INSERT INTO Products (ProductName, Price, Stock, CategoryID) VALUES

('Laptop', 1200, 50, 1),('Smartphone', 800, 150, 1), ('Table', 300, 20, 2),('Chair', 100, 100, 2),

('T-Shirt', 202, 200, 3), ('Novel', 150, 500, 4);

-- **Inserting into Customers**

INSERT INTO Customers (CustomerName, City, Email, PhoneNumber) VALUES

('Anil ', 'Kochi', 'anil@gmail.com', '9847012345'),

('Lakshmi', 'Thiruvananthapuram', 'lakshmi@gmail.com', '9447054321'),

('Sree', 'Pathanamthitta', 'sree@gmail.com', '9895067890'),

('Vishnu', 'Pathanamthitta', 'vishnu@gmail.com', '9746098765');

-- **Inserting into Orders**

INSERT INTO Orders (OrderDate, CustomerID, ProductID, Quantity) VALUES

('2024-08-01', 1, 1, 2),('2024-08-03', 2, 2, 3), ('2024-08-05', 3, 3, 1), ('2024-08-07', 4, 4, 5),

('2024-08-09', 1, 6, 10), ('2024-08-11', 2, 5, 7);

**List all orders along with the names of the customers who placed them. Include the order date and the product names in the result.**

SELECT o.OrderID , O.OrderDate , p.ProductName

FROM Orders o

JOIN Products p

ON o.ProductID = p.ProductID;

**Find all products that belong to the 'Electronics' category and list their names along with their prices and stock quantities.**

SELECT p.ProductName , p.Price ,p.Stock , c.CategoryName

FROM Products p

JOIN Categories c

ON p.CategoryID = c.CategoryID

WHERE c.CategoryName = 'Electronics'

**Retrieve total sales for each product category, including categories where total sales exceed 2000.**

SELECT p.CategoryID , SUM(o.Quantity \* p.Price)

FROM Orders o

INNER JOIN Products p

ON o.ProductID = p.ProductID

GROUP BY p.CategoryID

HAVING SUM(o.Quantity \* p.Price) > 2000

**Retrieve products in the 'Electronics' or 'Furniture' categories with prices between 100 and 1000, ordered by price descending.**

select p.ProductID , p.ProductName , c. CategoryName ,p.Price

FROM Products p

INNER JOIN Categories c

ON p. CategoryID = c.CategoryID

Where (c. CategoryName = 'Electronics' OR c. CategoryName = 'Furniture') AND (p.Price between 300 and 1000)

order by p.Price desc

SELECT \* FROM Orders

SELECT \* FROM Products

SELECT \* FROM Categories

SELECT \* FROM Customers

**Tasks**

**2) 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.**

use AuthenticationDB;

--TASK 1 and 2

-- CREATE TABLE

/\*

CREATE TABLE Signup (

UserID INT PRIMARY KEY IDENTITY(1,1),

Username VARCHAR(50) NOT NULL,

Email VARCHAR(50) UNIQUE NOT NULL,

User\_Password VARCHAR(100) NOT NULL,

Phone VARCHAR(20),

Gender VARCHAR(10),

DateOfBirth DATE,

StreetAddress VARCHAR(255),

City VARCHAR(100),

Country VARCHAR(100)

);

\*/

-- INSERT VALUES

INSERT INTO Signup (Username, Email, User\_Password, Phone, Gender, DateOfBirth, StreetAddress, City, Country)

VALUES

('Sreehari', 'sreehari@gmail.com', HASHBYTES('SHA2\_256', 'sree123'), '9562148689', 'Male', '1990-01-01', 'puthenvila', 'Kochi', 'India'),

('Amal', 'amal@gmail.com', HASHBYTES('SHA2\_256', 'amal123'), '9562148689', 'Male', '1988-05-15', 'newhouse', 'Thrissur', 'India'),

('Vishnu', 'vishnu@gmail.com', HASHBYTES('SHA2\_256', 'vishnu123'), '9562148689', 'Male', '1992-07-22', 'streetone', 'Calicut', 'India'),

('Bijo', 'bijo@gmail.com', HASHBYTES('SHA2\_256', 'bijo123'), '9562148689', 'Male', '1995-11-30', 'myhouse', 'Kannur', 'India');

UPDATE Signup

SET Phone = '968776423'

WHERE UserID = 4

DELETE FROM signup

WHERE UserID = 3;

--Login table

CREATE TABLE login (

LoginID INT PRIMARY KEY IDENTITY(1,1),

UserID INT FOREIGN KEY REFERENCES Signup(UserID),

LoginTime DATETIME DEFAULT GETDATE()

);

INSERT INTO login (UserID)

VALUES

(1), (2), (4);

**3) To create an employee table and how to achieve or get the second highest salary from the table.**

-- DB used EmployeeDB

CREATE TABLE employee (

EmployeeID INT PRIMARY KEY IDENTITY(1,1),

EmployeeName VARCHAR(50) ,

Department VARCHAR(50) ,

Salary DECIMAL(10, 2),

);

INSERT INTO employee (EmployeeName, Department, Salary)

VALUES

('Sree', 'IT', 75000.00),

('Vishnu', 'HR', 68000.00),

('Rahul', 'IT', 72000.00),

('Sneha', 'Sales', 65000.00),

('AARUN', 'Finance', 80000.00),

('Vikram', 'Sales', 62000.00),

('Pooja', 'IT', 73000.00);

-- second highest salary

SELECT MAX(Salary) as Second\_highest

FROM employee

WHERE Salary < (SELECT MAX(Salary) FROM employee);

**4) Perform the SQL query to list the number of employees in each department.**

SELECT Department , COUNT(EmployeeID) AS Number\_of\_Employees

FROM employee

GROUP BY Department

**5) To create two tables and implement all the SQL join concepts.**

-- create database PurchaseDB

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY IDENTITY(1,1),

CustomerName VARCHAR(100) ,

Email VARCHAR(100) UNIQUE,

City VARCHAR(50) ,

Phone VARCHAR(15)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY IDENTITY(1,1),

CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),

OrderDate DATETIME DEFAULT GETDATE(),

Amount DECIMAL(10, 2) );

INSERT INTO Customers (CustomerName, Email, City, Phone)

VALUES

('Arjun', 'arjun@gmail.com', 'Kochi', '9447000001'),

('Sree ', 'sree@gmail.com', 'Thiruvananthapuram', '9447000002'),

('Rajesh', 'rajesh.kumar@gmail.com', 'Kozhikode', '9447000003'),

('Sreelakshmi', 'sreelakshmi@gmail.com', 'Kochi', '9447000004'),

('Vijay', 'vijay@gmail.com', 'Kannur', '9447000005');

INSERT INTO Orders (CustomerID, OrderDate, Amount)

VALUES

(1, '2024-08-01', 150.00),

(1, '2024-08-15', 200.00),

(2, '2024-08-20', 300.00),

(3, '2024-08-22', 250.00),

(4, '2024-08-25', 100.00),

(1, '2024-08-26', 350.00);

**-- inner join**

SELECT

c.CustomerName, c.Email, o.OrderID, o.OrderDate, o.Amount

FROM

Customers c

INNER JOIN Orders o ON c.CustomerID = o.CustomerID;

**-- left join**

SELECT

c.CustomerName, c.Email, c.City, o.OrderID, o.OrderDate, o.Amount

FROM

Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID;

**-- right join**

SELECT

c.CustomerName, c.Email, o.OrderID, o.OrderDate, o.Amount

FROM

Customers c

RIGHT JOIN Orders o ON c.CustomerID = o.CustomerID;

**--FULL OUTER JOIN**

SELECT

c.CustomerName, c.Email, o.OrderID, o.OrderDate, o.Amount

FROM

Customers c

FULL OUTER JOIN Orders o ON c.CustomerID = o.CustomerID;

**--self join**

SELECT

a.CustomerName AS CustomerA,

b.CustomerName AS CustomerB,

a.City

FROM Customers a

INNER JOIN Customers b ON a.City = b.City

WHERE a.CustomerID <> b.CustomerID;

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

USE AuthenticationDB;

**--add data**

CREATE PROCEDURE InsertData

@Username VARCHAR(50),

@Email VARCHAR(50),

@Password VARCHAR(100),

@Phone VARCHAR(20),

@Gender VARCHAR(10),

@DateOfBirth DATE,

@StreetAddress VARCHAR(255),

@City VARCHAR(100),

@Country VARCHAR(100)

AS

BEGIN

INSERT INTO Signup (Username, Email, User\_Password, Phone, Gender, DateOfBirth, StreetAddress, City, Country)

VALUES ( @Username, @Email, HASHBYTES('SHA2\_256', @Password), @Phone, @Gender, @DateOfBirth, @StreetAddress,

@City, @Country );

END;

**-- view data**

CREATE PROCEDURE Get\_data

@Email VARCHAR(50)

AS

BEGIN

SELECT \* FROM Signup

WHERE Email = @Email

END;

**--update user**

CREATE PROCEDURE UpdateUser

@UserID INT,

@Username VARCHAR(50) = NULL,

@Email VARCHAR(50) = NULL,

@Password VARCHAR(100) = NULL,

@Phone VARCHAR(20) = NULL,

@Gender VARCHAR(10) = NULL,

@DateOfBirth DATE = NULL,

@StreetAddress VARCHAR(255) = NULL,

@City VARCHAR(100) = NULL,

@Country VARCHAR(100) = NULL

AS

BEGIN

UPDATE Signup

SET Username = COALESCE(@Username, Username),

Email = COALESCE(@Email, Email),

User\_Password = COALESCE(HASHBYTES('SHA2\_256' , @Password), User\_Password),

Phone = COALESCE(@Phone, Phone),

Gender = COALESCE(@Gender, Gender),

DateOfBirth = COALESCE(@DateOfBirth, DateOfBirth),

StreetAddress = COALESCE(@StreetAddress, StreetAddress),

City = COALESCE(@City, City),

Country = COALESCE(@Country, Country)

WHERE UserID = @UserID;

END;

**-- delete user**

CREATE PROCEDURE DeleteUser

@UserID INT

AS

BEGIN

DELETE FROM Signup

WHERE UserID = @UserID;

END;

**--Calling procedure**

EXECUTE InsertData

@Username = 'JohnDoe',

@Email = 'johndoe@example.com',

@Password = 'securepassword',

@Phone = '1234567890',

@Gender = 'Male',

@DateOfBirth = '1990-01-01',

@StreetAddress = '456 Elm St',

@City = 'CityName',

@Country = 'CountryName';

\*/

--EXECUTE Get\_data @Email = 'amal@gmail.com'

--EXECUTE UpdateUser @UserID = 1, @Phone = '0987654321', @City = 'Pta';

--EXEC DeleteUser @UserID = 5;

--select \* from Signup

**7) create a single stored procedure for implementing the CRUD operations on the Student Admission Form page.**

--AuthenticationDB

CREATE TABLE StudentAdmission (

StudentID INT IDENTITY(1,1) PRIMARY KEY,

Fullname VARCHAR(100),

Email VARCHAR(100) ,

Phone VARCHAR(15) ,

StudentAddress VARCHAR(200) ,

City VARCHAR(50) ,

DOB DATE ,

Gender VARCHAR(10) CHECK (Gender IN ('M', 'F' , 'O')),

Course VARCHAR(100)

);

CREATE PROCEDURE ManageStudents

@Type VARCHAR(20),

@StudentID INT = NULL ,

@Fullname VARCHAR(100) = NULL,

@Email VARCHAR(100) = NULL ,

@Phone VARCHAR(15) = NULL ,

@StudentAddress VARCHAR(200) = NULL ,

@City VARCHAR(50) = NULL,

@DOB DATE = NULL ,

@Gender VARCHAR(10) = NULL,

@Course VARCHAR(100) = NULL

AS

BEGIN

IF @Type = 'INSERT'

BEGIN

INSERT INTO StudentAdmission (Fullname , Email , Phone , StudentAddress , City , DOB , Gender , Course)

VALUES (@Fullname , @Email , @Phone , @StudentAddress , @City , @DOB , @Gender , @Course);

END

ELSE IF @Type = 'SHOW'

BEGIN

SELECT \* FROM StudentAdmission;

END

ELSE IF @Type = 'UPDATE'

BEGIN

UPDATE StudentAdmission

SET FullName = ISNULL(@FullName, FullName),

Email = ISNULL(@Email, Email),

Phone = ISNULL(@Phone, Phone),

StudentAddress = ISNULL(@StudentAddress, StudentAddress),

City = ISNULL(@City, City),

DOB = ISNULL(@DOB, DOB),

Gender = ISNULL(@Gender, Gender),

Course = ISNULL(@Course, Course)

WHERE StudentID = @StudentID;

END

ELSE IF @Type = 'DELETE'

BEGIN

DELETE FROM StudentAdmission

WHERE StudentID = @StudentID;

END

END

**-- insert data**

EXEC ManageStudents @Type = 'INSERT',

@Fullname = 'sree',

@Email = 'sreeh@gmail.com',

@Phone = '9447088001',

@StudentAddress = 'nehouse',

@City = 'Kochi',

@DOB = '2000-11-01',

@Gender = 'M',

@Course = 'MCA';

**-- update data**

EXEC ManageStudents @Type = 'UPDATE',

@StudentID = 1,

@Phone = '9447000002',

@StudentAddress = 'newvilla';

**-- delete data**

EXEC ManageStudents @Type = 'DELETE',

@StudentID = 1;

**-- view data**

--EXEC ManageStudents @Type = 'SHOW';

**8 ) To Learn and implementing the example queries or the following tasks: a. Normalization Techniques - All the normal forms b. Indexing - Cluster and non-cluster indexing c. Pivot and unpivot the values in SQL Table d. Merge concepts in SQL Table.**

**NORMAL FORMS**

--create database NormalformDB

**-- 1NF**

CREATE TABLE Employees (

EMP\_ID INT PRIMARY KEY,

Fullname VARCHAR(100),

Place VARCHAR(50)

);

CREATE TABLE EmployeePhones (

Phone\_ID INT PRIMARY KEY IDENTITY(1,1),

EMP\_ID INT FOREIGN KEY REFERENCES Employees(EMP\_ID),

Phone VARCHAR(15)

);

INSERT INTO Employees (EMP\_ID, Fullname, Place)

VALUES

(1, 'Amit Sharma', 'Delhi'),

(2, 'Rajesh Kumar', 'Mumbai'),

(3, 'Priya Singh', 'Chennai'),

(4, 'Sunita Menon', 'Bangalore'),

(5, 'Vijay Patel', 'Ahmedabad');

INSERT INTO EmployeePhones (EMP\_ID, Phone)

VALUES

(1, '9876543210'), (1, '9123456789') (2, '8765432109'),(3, '9988776655'),

(4, '8899776655'), (5, '7890654321'), (5, '7012345678');

select \* from EmployeePhones

**--2NF**

-- StudentID StudentName CourseID CourseName Instructor

/\*

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

StudentName VARCHAR(50) NOT NULL

);

CREATE TABLE Courses (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(50) NOT NULL,

Instructor VARCHAR(50) NOT NULL

);

CREATE TABLE Enrollments (

StudentID INT,

CourseID INT,

PRIMARY KEY (StudentID, CourseID),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

INSERT INTO Students (StudentID, StudentName)

VALUES

(1, 'sree'),

(2, 'vishnu');

INSERT INTO Courses (CourseID, CourseName, Instructor)

VALUES

(101, 'Math', 'Mr. Smith'),

(102, 'English', 'Mr. Arun');

INSERT INTO Enrollments (StudentID, CourseID)

VALUES

(1, 101),

(1, 102),

(2, 101);

**-- 3NF**

--EMP\_ID EMP\_NAME EMP\_ZIP EMP\_STATE EMP\_CITY

CREATE TABLE employee\_zip (

emp\_zip VARCHAR(10) PRIMARY KEY,

emp\_state VARCHAR(50),

emp\_city VARCHAR(50)

);

CREATE TABLE employee (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

emp\_zip VARCHAR(10) FOREIGN KEY REFERENCES employee\_zip(emp\_zip)

);

INSERT INTO employee\_zip (emp\_zip, emp\_state, emp\_city) VALUES

('695014', 'kerala', 'trivandrum'),

('678001', 'kerala', 'palakkad'),

('682001', 'kerala', 'ernakulam'),

('683104', 'kerala', 'kottayam'),

('673001', 'kerala', 'kozhikode');

INSERT INTO employee (emp\_id, emp\_name, emp\_zip) VALUES

(1, 'arjun', '695014'),

(2, 'manju', '678001'),

(3, 'deepak', '682001'),

(4, 'aishwarya', '683104'),

(5, 'vivek', '673001');

**--Indexing - Cluster and non-cluster indexing**

CREATE TABLE PersonalInfo (

ID INT PRIMARY KEY,

Fullname VARCHAR(100),

Salary VARCHAR(50)

);

INSERT INTO PersonalInfo (ID, Fullname, Salary)

VALUES

(1, 'sree ', '50000'), (2, ' vishnu', '60000'), (3, 'stephin', '55000'), (4, 'tony', '65000');

-- sp\_helpindex PersonalInfo;

--CREATE CLUSTERED INDEX IX\_PersonalInfo\_Salary ON PersonalInfo(Salary DESC);

--select \* from PersonalInfo;

**-- merge concept**

CREATE TABLE TargetProduct (

ProductID INT PRIMARY KEY,

Price DECIMAL(10, 2),

Name VARCHAR(100)

);

CREATE TABLE SourceProduct (

ProductID INT PRIMARY KEY,

Price DECIMAL(10, 2),

Name VARCHAR(100) );

INSERT INTO TargetProduct (ProductID, Price, Name)

VALUES

(1, 300, 'table'), (2, 340, ' chair'), (3, 320, 'bag');

INSERT INTO SourceProduct (ProductID, Price, Name)

VALUES (2, 400, 'chair'), (3, 320, 'bag'), (4, 100, 'book');

select \* from SourceProduct

select \* from TargetProduc

MERGE TargetProduct AS t

USING SourceProduct AS s

ON t.ProductID = s.ProductID

WHEN MATCHED THEN UPDATE

SET t.Price =s.Price , t.ProductName =s.ProductName

WHEN NOT MATCHED BY TARGET THEN

INSERT VALUES (s.ProductID , s.Price , s.ProductName);