SQL Assignment 2

1.For an online purchasing database, create entity relationship diagrams. Create a database object from your entity diagram.

Create the Database Object:  
CREATE DATABASE OnlineStore;

USE OnlineStore;

-- Customer Table

CREATE TABLE Customer (

CustomerID INT AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Email VARCHAR(100),

Phone VARCHAR(20),

Address VARCHAR(255));

-- Category Table

CREATE TABLE Category (

CategoryID INT AUTO\_INCREMENT PRIMARY KEY,

CategoryName VARCHAR(100));

-- Product Table

CREATE TABLE Product (

ProductID INT AUTO\_INCREMENT PRIMARY KEY,

ProductName VARCHAR(100),

Description TEXT,

Price DECIMAL(10, 2),

CategoryID INT,

FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID));

-- Order Table

CREATE TABLE Orders (

OrderID INT AUTO\_INCREMENT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

TotalAmount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID));

-- OrderItem Table

CREATE TABLE OrderItem (

OrderItemID INT AUTO\_INCREMENT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

UnitPrice DECIMAL(10, 2),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Product(ProductID));

-- Payment Table

CREATE TABLE Payment (

PaymentID INT AUTO\_INCREMENT PRIMARY KEY,

OrderID INT,

PaymentMethod VARCHAR(50),

PaymentDate DATE,

Amount DECIMAL(10, 2),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID));

-- Shipping Table

CREATE TABLE Shipping (

ShippingID INT AUTO\_INCREMENT PRIMARY KEY,

OrderID INT,

ShippingAddress VARCHAR(255),

ShippingMethod VARCHAR(50),

ShippingDate DATE,

DeliveryDate DATE,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID));

2.Create a SQL store process to register the use of the database, complete it with proper validation and transaction rollback and commit.

Stored Procedure for User Registration:  
DELIMITER $$

CREATE PROCEDURE RegisterUser(

IN p\_FirstName VARCHAR(50),

IN p\_LastName VARCHAR(50),

IN p\_Email VARCHAR(100),

IN p\_Phone VARCHAR(20),

IN p\_Address VARCHAR(255))

BEGIN  
 -- Declare variables to handle errors and duplicate checks  
 DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

-- Rollback the transaction in case of an error

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'An error occurred during registration.';

END;

-- Start transaction

START TRANSACTION;

-- Check if the email already exists in the Customer table

IF EXISTS (SELECT 1 FROM Customer WHERE Email = p\_Email) THEN

-- Rollback transaction if email is already taken

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Email already exists. Please use a different email.';

ELSE

-- Insert the new user into the Customer table if email is unique

INSERT INTO Customer (FirstName, LastName, Email, Phone, Address)

VALUES (p\_FirstName, p\_LastName, p\_Email, p\_Phone, p\_Address);

-- Commit the transaction if the insert is successful

COMMIT;

END IF;

END$$

DELIMITER ;

3.List the SQL aggregate function and demonstrate how to utilize it.

SQL **aggregate functions** are used to perform calculations on a set of values and return a single value. They are often used with the GROUP BY clause to group rows that share a common value. Here's a list of commonly used aggregate functions:

**COUNT()**: Returns the number of rows, **SUM()**: Returns the sum of a numeric column.

**AVG()**: Returns the average value of a numeric column , **MIN()**: Returns the minimum value.  
 **MAX()**: Returns the maximum value.  
SELECT COUNT(\*) AS TotalOrders FROM Sales;  
SELECT SUM(Amount) AS TotalSales FROM Sales;  
SELECT AVG(Amount) AS AverageSales FROM Sales;  
SELECT MIN(Amount) AS MinSale FROM Sales;  
SELECT MAX(Amount) AS MaxSale FROM Sales;  
SELECT CustomerID, SUM(Amount) AS TotalSales FROM Sales GROUP BY CustomerID;  
SELECT CustomerID, SUM(Amount) AS TotalSales FROM Sales GROUP BY CustomerID HAVING SUM(Amount) > 500;  
  
4.In SQL, create a pivot query.  
In SQL, a **pivot query** is used to transform data from rows into columns, essentially "pivoting" the table. This is often done by aggregating data based on a certain field and displaying those aggregates as columns.  
SELECT

CustomerID,

SUM(CASE WHEN Product = 'Laptop' THEN Amount ELSE 0 END) AS Laptop,

SUM(CASE WHEN Product = 'Mobile' THEN Amount ELSE 0 END) AS Mobile,

SUM(CASE WHEN Product = 'Tablet' THEN Amount ELSE 0 END) AS Tablet

FROM Sales

GROUP BY CustomerID;

5.With an example, describe how to join in SQL.  
  
In SQL, **JOIN** is used to combine rows from two or more tables based on a related column between them. There are different types of joins, each with its own behavior regarding how it combines the data.

**Types of SQL Joins:**

1. **INNER JOIN**: Returns only the rows where there is a match in both tables.
2. **LEFT (OUTER) JOIN**: Returns all rows from the left table, and the matched rows from the right table. If there's no match, NULL values are returned for columns from the right table.
3. **RIGHT (OUTER) JOIN**: Similar to LEFT JOIN but returns all rows from the right table, with NULLs for non-matching rows from the left table.
4. **FULL (OUTER) JOIN**: Returns all rows when there is a match in either left or right table. If there is no match, NULL values are returned.
5. **CROSS JOIN**: Returns the Cartesian product of the two tables. Each row from the first table is combined with all rows from the second table.
6. **SELF JOIN**: A table is joined with itself.

6.How to locate the 4th highest value in a column in a row. Create your table.

To locate the 4th highest value in a column in SQL, we can use the ORDER BY clause along with the LIMIT or OFFSET clauses, depending on the SQL dialect you're using

CREATE TABLE Employees ( EmployeeID INT PRIMARY KEY, EmployeeName VARCHAR(50), Salary DECIMAL(10, 2) );   
-- Insert some sample data INSERT INTO Employees (EmployeeID, EmployeeName, Salary)   
VALUES (1, 'John', 5000), (2, 'Jane', 6000), (3, 'Alice', 7000), (4, 'Bob', 8000), (5, 'Charlie', 7500), (6, 'David', 9000), (7, 'Eva', 8500);  
  
SELECT Salary FROM Employees ORDER BY Salary DESC LIMIT 1 OFFSET 3;