Course Code	:	**BCSL-034**
Title	:	**DBMS Lab**
Assignment Number	:	**BCA(III)/L-034/Assignment/2024-25**
Maximum Marks	:	**50**
Weightage	:	**25%**
Last Date of Submission	:	**31st October,2024(for July Session)** **30th April,2025(for January Session)**

This assignment has only one question. Answer the question. This question carries 40 marks. Rest 10 marks are for viva voce. You may use illustrations and diagrams to enhance the explanation. Assumptions can be made wherever required. Please go through the guidelines regarding the assignments given in the programme guide for the format of presentation.

Q1.

Design and implement a simple database using MS-Access for an O***nline Retail Store***, using fundamental concepts of database management systems such as creating tables, establishing relationships, performing CRUD (Create, Read, Update, Delete) operations, and writing basic SQL queries.

Create a database schema for an online retail store and implement various operations on it. The database will manage information about customers, products, orders, and order details. Use SQL to create tables, establish relationships, and perform queries to manipulate and retrieve data.

I. **Create Database Schema:** **(15 Marks)**

- **Customers Table:**
  - **customer_id** (Primary Key, INT, Auto Increment)
  - **first_name** (VARCHAR)
  - **last_name** (VARCHAR)
  - **email** (VARCHAR, Unique)
  - **phone** (VARCHAR)
  - **address** (VARCHAR)
- **Products Table:**
  - **product_id** (Primary Key, INT, Auto Increment)
  - **product_name** (VARCHAR)
  - **description** (TEXT)
  - **price** (DECIMAL)
  - **stock_quantity** (INT)
- **Orders Table:**
  - **order_id** (Primary Key, INT, Auto Increment)
  - **customer_id** (Foreign Key, INT)
  - **order_date** (DATE)
  - **status** (VARCHAR)
- **OrderDetails Table:**
  - **order_detail_id** (Primary Key, INT, Auto Increment)
  - **order_id** (Foreign Key, INT)
  - **product_id** (Foreign Key, INT)
  - **quantity** (INT)
  - **total_price** (DECIMAL)

II. **Relationships:**

- Each order is placed by one customer.
- Each order can have multiple products.

- Each product can be part of multiple orders.

Draw an ER-diagram for this application. **(5 Marks)**

III. **Operations:**

- **CRUD (Create, Read, Update, Delete) Operations** **(7 ½ Marks)**
  - Insert new records into each table.
  - Read/display records from each table.
  - Update existing records in each table.
  - Delete records from each table.

IV. Write and execute the following SQL Queries: **(12 ½ Marks)**

1. Retrieve all orders along with the customer details who placed the order.
2. Find all products that have been ordered by a specific customer.
3. Retrieve the total sales for each product.
4. Find all customers who have placed at least one order.
5. Retrieve the total quantity of products ordered by each customer.
6. Find all orders and their order details for a specific customer.
7. Retrieve all products along with the total quantity ordered.
8. Find the total revenue generated from orders placed within a specific date range.
9. Retrieve all customers who have ordered a specific product.
10. Find the most frequently ordered product.
11. Retrieve the average order value for each customer.
12. Find all products that have never been ordered.
13. Retrieve the total number of orders placed each month.
14. Retrieve the total number of products ordered in each order.
15. Find the top 5 customers based on total spending.
16. Retrieve all orders placed on a specific date.
17. Find the total number of unique products ordered by each customer.
18. Retrieve the order details for the order with the highest total price.
19. Find the top 3 products based on the total quantity ordered.
20. Retrieve the total sales for each month.

Ans

I. Create Database Schema

1. **Customers Table**

```
CREATE TABLE Customers (
    customer_id AUTOINCREMENT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(15),
    address VARCHAR(255)
);
```

2. **Products Table**

```
CREATE TABLE Products (
    product_id AUTOINCREMENT PRIMARY KEY,
    product_name VARCHAR(100),
    description TEXT,
    price DECIMAL(10, 2),
    stock_quantity INT
);
```

3. **Orders Table**

```
CREATE TABLE Orders (
    order_id AUTOINCREMENT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    status VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

4. **OrderDetails Table**

```
CREATE TABLE OrderDetails (
    order_detail_id AUTOINCREMENT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT,
    total_price DECIMAL(10, 2),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

II. Relationships

- **ER-Diagram**: The ER-Diagram should show the relationships between the tables.
    - **Customers** are linked to **Orders** (one-to-many).
    - **Orders** are linked to **OrderDetails** (one-to-many).
    - **Products** are linked to **OrderDetails** (many-to-many, resolved by the OrderDetails table).

III. Operations: CRUD Operations

1. **Insert Records**

```
-- Insert into Customers
INSERT INTO Customers (first_name, last_name, email, phone, address)
VALUES ('John', 'Doe', 'john.doe@example.com', '123-456-7890', '123 Elm St.');

-- Insert into Products
INSERT INTO Products (product_name, description, price, stock_quantity)
VALUES ('Laptop', 'High-performance laptop', 1200.00, 50);

-- Insert into Orders
INSERT INTO Orders (customer_id, order_date, status)
VALUES (1, #2024-09-01#, 'Shipped');

-- Insert into OrderDetails
INSERT INTO OrderDetails (order_id, product_id, quantity, total_price)
VALUES (1, 1, 2, 2400.00);
```

2. **Read/Display Records**

```sql
SELECT * FROM Customers;
SELECT * FROM Products;
SELECT * FROM Orders;
SELECT * FROM OrderDetails;
```

3. **Update Records**

```sql
UPDATE Products
SET price = 1150.00
WHERE product_id = 1;
```

4. **Delete Records**

```sql
DELETE FROM Customers
WHERE customer_id = 1;
```

IV. SQL Queries

1. **Retrieve all orders along with the customer details who placed the order**

```sql
SELECT Orders.order_id, Orders.order_date, Customers.first_name, Customers.last_name, Customers.email
FROM Orders
JOIN Customers ON Orders.customer_id = Customers.customer_id;
```

2. **Find all products that have been ordered by a specific customer**

```sql
SELECT Products.product_name, OrderDetails.quantity
FROM OrderDetails
JOIN Orders ON OrderDetails.order_id = Orders.order_id
JOIN Products ON OrderDetails.product_id = Products.product_id
WHERE Orders.customer_id = 1;
```

3. **Retrieve the total sales for each product**

```sql
SELECT Products.product_name, SUM(OrderDetails.total_price) AS total_sales
FROM OrderDetails
JOIN Products ON OrderDetails.product_id = Products.product_id
GROUP BY Products.product_name;
```

4. **Find all customers who have placed at least one order**

```sql
SELECT DISTINCT Customers.first_name, Customers.last_name, Customers.email
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id;
```

5. **Retrieve the total quantity of products ordered by each customer**

```sql
SELECT Customers.first_name, Customers.last_name, SUM(OrderDetails.quantity) AS total_quantity
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
GROUP BY Customers.first_name, Customers.last_name;
```

6. **Find all orders and their order details for a specific customer**

```sql
SELECT Orders.order_id, Orders.order_date, Products.product_name, OrderDetails.quantity, OrderDetails.total_price
FROM Orders
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
JOIN Products ON OrderDetails.product_id = Products.product_id
WHERE Orders.customer_id = 1;
```

7. **Retrieve all products along with the total quantity ordered**

```sql
SELECT Products.product_name, SUM(OrderDetails.quantity) AS total_quantity_ordered
FROM Products
JOIN OrderDetails ON Products.product_id = OrderDetails.product_id
GROUP BY Products.product_name;
```

8. **Find the total revenue generated from orders placed within a specific date range**

```sql
SELECT SUM(OrderDetails.total_price) AS total_revenue
FROM Orders
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
WHERE Orders.order_date BETWEEN #2024-01-01# AND #2024-12-31#;
```

9. **Retrieve all customers who have ordered a specific product**

```sql
SELECT DISTINCT Customers.first_name, Customers.last_name
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
WHERE OrderDetails.product_id = 1;
```

10. **Find the most frequently ordered product**

```sql
SELECT Products.product_name, SUM(OrderDetails.quantity) AS total_quantity_ordered
FROM Products
JOIN OrderDetails ON Products.product_id = OrderDetails.product_id
GROUP BY Products.product_name
ORDER BY total_quantity_ordered DESC
LIMIT 1;
```

11. **Retrieve the average order value for each customer**

```sql
SELECT Customers.first_name, Customers.last_name, AVG(OrderDetails.total_price) AS average_order_value
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
GROUP BY Customers.first_name, Customers.last_name;
```

12. **Find all products that have never been ordered**

```sql
SELECT Products.product_name
FROM Products
LEFT JOIN OrderDetails ON Products.product_id = OrderDetails.product_id
WHERE OrderDetails.product_id IS NULL;
```

13. **Retrieve the total number of orders placed each month**

```sql
SELECT YEAR(order_date) AS order_year, MONTH(order_date) AS order_month, COUNT(*) AS total_orders
FROM Orders
GROUP BY YEAR(order_date), MONTH(order_date);
```

14. **Retrieve the total number of products ordered in each order**

```sql
SELECT Orders.order_id, SUM(OrderDetails.quantity) AS total_products
FROM Orders
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
GROUP BY Orders.order_id;
```

15. **Find the top 5 customers based on total spending**

```sql
SELECT Customers.first_name, Customers.last_name, SUM(OrderDetails.total_price) AS total_spending
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
GROUP BY Customers.first_name, Customers.last_name
ORDER BY total_spending DESC
LIMIT 5;
```

16. **Retrieve all orders placed on a specific date**

```sql
SELECT Orders.order_id, Orders.order_date, Customers.first_name, Customers.last_name
FROM Orders
JOIN Customers ON Orders.customer_id = Customers.customer_id
WHERE Orders.order_date = #2024-09-01#;
```

17. **Find the total number of unique products ordered by each customer**

```sql
SELECT Customers.first_name, Customers.last_name, COUNT(DISTINCT OrderDetails.product_id) AS unique_products
FROM Customers
JOIN Orders ON Customers.customer_id = Orders.customer_id
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
GROUP BY Customers.first_name, Customers.last_name;
```

18. **Retrieve the order details for the order with the highest total price**

```sql
SELECT Orders.order_id, Products.product_name, OrderDetails.quantity, OrderDetails.total_price
FROM Orders
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
JOIN Products ON OrderDetails.product_id = Products.product_id
WHERE Orders.order_id = (
    SELECT Orders.order_id
    FROM Orders
    JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
    GROUP BY Orders.order_id
    ORDER BY SUM(OrderDetails.total_price) DESC
    LIMIT 1
);
```

19. **Find the top 3 products based on the total quantity ordered**

```sql
SELECT Products.product_name, SUM(OrderDetails.quantity) AS total_quantity_ordered
FROM Products
JOIN OrderDetails ON Products.product_id = OrderDetails.product_id
GROUP BY Products.product_name
ORDER BY total_quantity_ordered DESC
LIMIT 3;
```

20. **Retrieve the total sales for each month**

```sql
SELECT YEAR(Orders.order_date) AS order_year, MONTH(Orders.order_date) AS order_month, SUM(OrderDetails.total_price) AS total_sales
FROM Orders
JOIN OrderDetails ON Orders.order_id = OrderDetails.order_id
GROUP BY YEAR(Orders.order_date), MONTH(Orders.order_date);
```