SQL Server Assessment

Your company **TechMart** is developing a small database to manage:

- Customers
- Orders
- Products
- Employees
- Payments

You are required to write SQL queries, create integrity rules, and implement logics using functions, procedures, and triggers.

Database Schema

1. Customers

```
Customers(
   CustID INT PRIMARY KEY,
   CustName VARCHAR(100),
   Email VARCHAR(200),
   City VARCHAR(100)
)
```

2. Products

```
Products(
   ProductID INT PRIMARY KEY,
   ProductName VARCHAR(100),
   Price DECIMAL(10,2),
   Stock INT CHECK(Stock >= 0)
)
```

3. Orders

```
Orders(
   OrderID INT PRIMARY KEY,
   CustID INT FOREIGN KEY REFERENCES Customers(CustID),
   OrderDate DATE,
   Status VARCHAR(20)
)
```

4. OrderDetails

```
OrderDetails(
   DetailID INT PRIMARY KEY,
   OrderID INT FOREIGN KEY REFERENCES Orders(OrderID),
   ProductID INT FOREIGN KEY REFERENCES Products(ProductID),
   Qty INT CHECK(Qty > 0)
)
```

5. Payments

```
Payments(
   PaymentID INT PRIMARY KEY,
   OrderID INT FOREIGN KEY REFERENCES Orders(OrderID),
   Amount DECIMAL(10,2),
   PaymentDate DATE
)
```

SQL Queries

Q1. List customers who placed an order in the last 30 days.

(Use joins)

Q2. Display top 3 products that generated the highest total sales amount.

(Use aggregate + joins)

Q3. For each city, show number of customers and total order count.

Q4. Retrieve orders that contain more than 2 different products.

Q5. Show orders where total payable amount is greater than 10,000.

(Hint: SUM(Qty * Price))

Q6. List customers who ordered the same product more than once.

Q7. Display employee-wise order processing details

(Assume Orders table has EmployeeID column)

Views

1. Create a view vw_LowStockProducts

Show only products with stock < 5.
View should be **WITH SCHEMABINDING and Encrypted**

Functions

1. Create a table-valued function: fn_GetCustomerOrderHistory(@CustID)

Return: OrderID, OrderDate, TotalAmount.

2. Create a function fn_GetCustomerLevel(@CustID)

Logic:

- Total purchase > 1,00,000 → "Platinum"
- 50,000–1,00,000 → "Gold"
- Else → "Silver"

Procedures

1. Create a stored procedure to update product price

Rules:

- Old price must be logged in a PriceHistory table
- New price must be > 0
- If invalid, throw custom error.

2. Create a procedure sp_SearchOrders

Search orders by:

- Customer Name
- City
- Product Name
- Date range
  (Any parameter can be NULL → Dynamic WHERE)

Triggers

1. Create a trigger on Products

Prevent deletion of a product if it is part of any OrderDetails.

2. Create an AFTER UPDATE trigger on Payments

Log old and new payment values into a PaymentAudit table.

3. Create an INSTEAD OF DELETE trigger on Customers

Logic:

- If customer has orders → mark status as “Inactive” instead of deleting
- If no orders → allow deletion