Day 1: E-Commerce SQL Practice

Scenario Description

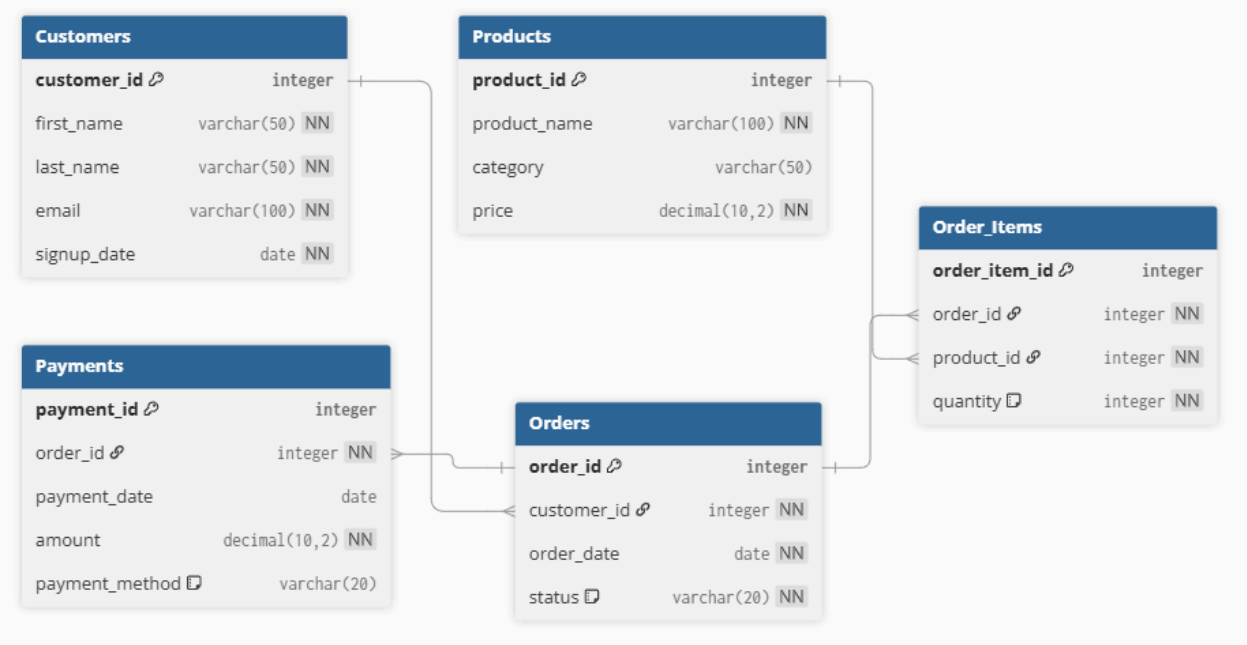
Business Context:

You are working for an online e-commerce platform. The platform tracks customers, their orders, products, and payments. The data is used to analyze customer behavior, sales trends, product performance, and financial reporting.

Why it matters:

- Understanding which products are bestsellers helps in inventory management.
- Analyzing customer orders helps in targeted marketing.
- Tracking payments ensures accurate revenue reporting.
- Cleaning and handling missing data ensures accurate analytics.

Database Schema



Tables

1. **Customers**

```
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    signup_date DATE NOT NULL
);
```

1. **Products**

```
CREATE TABLE Products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100) NOT NULL,
    category VARCHAR(50),
    price DECIMAL(10,2) NOT NULL
);
```

1. **Orders**

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT NOT NULL,
    order_date DATE NOT NULL,
    status VARCHAR(20) NOT NULL CHECK(status IN ('Pending','Shipped','Deli
vered','Cancelled')),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

1. **Order_Items**

```
CREATE TABLE Order_Items (
    order_item_id INT PRIMARY KEY,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL CHECK(quantity > 0),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

1. **Payments**

```
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY,
    order_id INT NOT NULL,
    payment_date DATE,
    amount DECIMAL(10,2) NOT NULL,
    payment_method VARCHAR(20) CHECK(payment_method IN ('Credit Car
d','Debit Card','UPI','Cash On Delivery')),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
```

Sample Data

Customers

```
INSERT INTO Customers VALUES
(1,'John','Doe','john.doe@example.com','2025-01-15'),
(2,'Jane','Smith','jane.smith@example.com','2025-02-10'),
(3,'Alice','Johnson','alice.j@example.com','2025-03-05');
```

Products

```
INSERT INTO Products VALUES
(101,'Laptop','Electronics',1200.00),
(102,'Smartphone','Electronics',800.00),
(103,'Headphones','Accessories',150.00),
(104,'Office Chair','Furniture',300.00);
```

Orders

```
INSERT INTO Orders VALUES
(1001,1,'2025-04-01','Delivered'),
(1002,2,'2025-04-03','Shipped'),
(1003,1,'2025-04-05','Pending');
```

Order_Items

```
INSERT INTO Order_Items VALUES
(1,1001,101,1),
(2,1001,103,2),
(3,1002,102,1),
(4,1003,104,1);
```

Payments

```
INSERT INTO Payments VALUES
(5001,1001,'2025-04-02',1500.00,'Credit Card'),
(5002,1002,'2025-04-04',800.00,'UPI');
```

ERD (Textual)

```
Customers (1) ──< Orders (M)
Orders (1) ──< Order_Items (M) >── Products (1)
Orders (1) ──< Payments (M)
```

Relationships:

- **Customers → Orders** : 1:M
- **Orders → Order_Items** : 1:M
- **Order_Items → Products** : M:1
- **Orders → Payments** : 1:M

SQL Questions

Easy

1. List all customers with their signup date.

Medium

1. Find the total amount spent by each customer (consider only delivered orders).

Hard

1. Retrieve the top 2 products by total quantity sold.

Difficult

1. Find customers who have orders but **no payments recorded** yet.

Expert

1. Identify the category with the highest revenue from delivered orders, and show total revenue per category.

Solutions with Explanations

Easy

```
SELECT first_name, last_name, signup_date
FROM Customers;
```

Explanation:

- Simple SELECT query fetching basic customer info.
- No joins or filters needed.

Medium

```
SELECT c.customer_id, c.first_name, c.last_name, SUM(oi.quantity * p.price)
AS total_spent
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
JOIN Order_Items oi ON o.order_id = oi.order_id
JOIN Products p ON oi.product_id = p.product_id
WHERE o.status = 'Delivered'
GROUP BY c.customer_id, c.first_name, c.last_name;
```

Explanation:

- Join Customers → Orders → Order_Items → Products.
- Multiply quantity by price for total per item.
- Filter only `Delivered` orders.
- Aggregate per customer using `SUM` and `GROUP BY`.

Tip: Add indexes on `Orders.customer_id` and `Order_Items.order_id` for performance.

Hard

```
SELECT p.product_name, SUM(oi.quantity) AS total_quantity
FROM Order_Items oi
JOIN Products p ON oi.product_id = p.product_id
GROUP BY p.product_id, p.product_name
ORDER BY total_quantity DESC
LIMIT 2;
```

Explanation:

- Sum quantity per product.
- Order by descending quantity to get bestsellers.
- `LIMIT 2` gives top 2.

Difficult

```
SELECT c.customer_id, c.first_name, c.last_name
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
LEFT JOIN Payments p ON o.order_id = p.order_id
WHERE p.payment_id IS NULL;
```

Explanation:

- Use `LEFT JOIN` to include all orders.
- `WHERE p.payment_id IS NULL` finds orders without payments.
- Useful for finance tracking.

Expert

```
SELECT p.category, SUM(oi.quantity * p.price) AS total_revenue
FROM Orders o
JOIN Order_Items oi ON o.order_id = oi.order_id
```

```
JOIN Products p ON oi.product_id = p.product_id
WHERE o.status = 'Delivered'
GROUP BY p.category
ORDER BY total_revenue DESC
LIMIT 1;
```

Explanation:

- Aggregate revenue by category for delivered orders.
- Helps identify high-performing categories.
- `SUM(oi.quantity * p.price)` calculates revenue per category.
- `LIMIT 1` finds the top category.

Tip: Index `Order_Items.product_id` and `Orders.status` to improve performance.