# SQL Project – Retail Store

Instructions to Learners:

1. Use the provided DDLs to create the following tables
    a. Customers
    b. Products
    c. Orders
    d. Order Items
    e. Payments
2. Use the provided DMLs to insert data into the tables created
3. Construct SQL queries to answer the given business questions

# TABLE STRUCTURES

Customers

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| customer_id | INT | PRIMARY KEY, AUTO_INCREMENT | Unique ID for each customer |
| name | VARCHAR(100) | NOT NULL | Full name of the customer |
| email | VARCHAR(100) | UNIQUE, NOT NULL | Unique email address |
| phone | VARCHAR(15) | | Optional phone number |
| created_at | DATETIME | DEFAULT CURRENT_TIMESTAMP | Account creation timestamp |

Products

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| product_id | INT | PRIMARY KEY, AUTO_INCREMENT | Unique product ID |
| name | VARCHAR(100) | NOT NULL | Name of the product |
| category | VARCHAR(50) | NOT NULL | Product category (e.g., Electronics) |
| price | DECIMAL(10,2) | NOT NULL CHECK (price > 0) | Product price |
| stock_quantity | INT | NOT NULL DEFAULT 0 | Units available in stock |
| added_on | DATE | DEFAULT CURRENT_TIMESTAMP | Date product was added |

Orders

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| order_id | INT | PRIMARY KEY, AUTO_INCREMENT | Unique order ID |
| customer_id | INT | FOREIGN KEY REFERENCES customers(customer_id) | Who placed the order |
| order_date | DATETIME | DEFAULT CURRENT_TIMESTAMP | When the order was placed |
| status | VARCHAR(20) | DEFAULT 'Pending' | Order status (Pending, Shipped, etc) |
| total_amount | DECIMAL(10,2) | | Calculated total |

Order Items

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| order_item_id | INT | PRIMARY KEY, AUTO_INCREMENT | Unique ID for each line item |
| order_id | INT | FOREIGN KEY REFERENCES orders(order_id) | Order to which item belongs |
| product_id | INT | FOREIGN KEY REFERENCES products(product_id) | Which product was bought |
| quantity | INT | NOT NULL CHECK (quantity > 0) | Quantity ordered |
| item_price | DECIMAL(10,2) | NOT NULL | Snapshot of product price at order |

Payments

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| payment_id | INT | PRIMARY KEY, AUTO_INCREMENT | Unique ID for payment |
| order_id | INT | FOREIGN KEY REFERENCES orders(order_id) | Which order was paid for |
| payment_date | DATETIME | DEFAULT CURRENT_TIMESTAMP | When payment was made |
| amount_paid | DECIMAL(10,2) | NOT NULL CHECK (amount_paid > 0) | Amount paid |
| method | VARCHAR(20) | NOT NULL | Method (Credit Card, UPI, etc.) |

Product Reviews

| Column Name | Data Type | Constraints | Description |
|---|---|---|---|
| review_id | INT | PRIMARY KEY, AUTO_INCREMENT | Unique ID for review |
| product_id | INT | FOREIGN KEY REFERENCES products(product_id) | Which product is reviewed |
| customer_id | INT | FOREIGN KEY REFERENCES customers(customer_id) | Who gave the review |
| rating | INT | NOT NULL CHECK (rating BETWEEN 1 AND 5) | Star rating |
| review_text | TEXT | | Review message |
| review_date | DATE | DEFAULT CURRENT_TIMESTAMP | When the review was posted |

# DDL SCRIPTS

**Customers**

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE,
    phone VARCHAR(15),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

**Products**

```
CREATE TABLE products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    category VARCHAR(50) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    stock_quantity INT NOT NULL DEFAULT 0,
    added_on DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

**Orders**

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    customer_id INT,
    order_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    status VARCHAR(20) DEFAULT 'Pending',
    total_amount DECIMAL(10,2),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

**Order Items**

```
CREATE TABLE order_items (
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    product_id INT,
    quantity INT NOT NULL CHECK (quantity > 0),
    item_price DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(order_id),
    FOREIGN KEY (product_id) REFERENCES products(product_id)
);
```

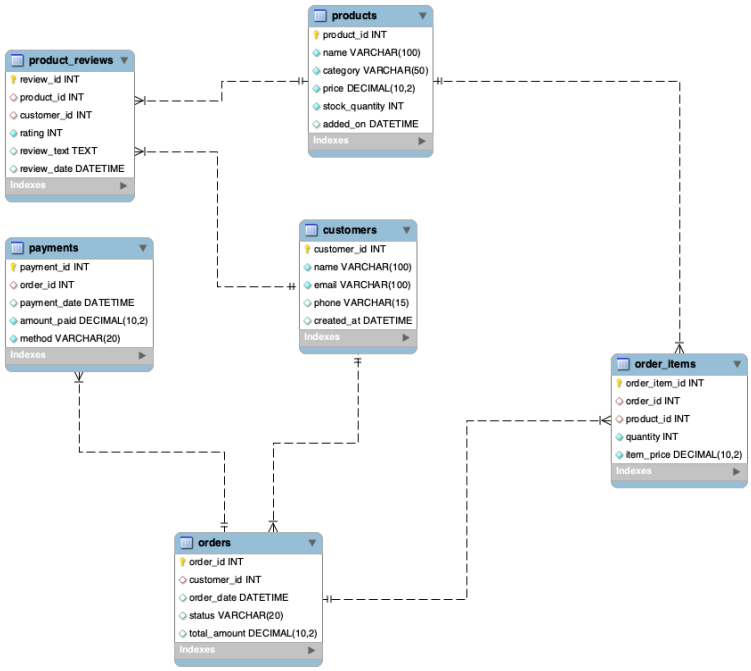
**Payments**

```
CREATE TABLE payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT,
    payment_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    amount_paid DECIMAL(10,2) NOT NULL CHECK (amount_paid > 0),
    method VARCHAR(20) NOT NULL,
    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);
```

### Product Reviews

```
CREATE TABLE product_reviews (
    review_id INT PRIMARY KEY AUTO_INCREMENT,
    product_id INT,
    customer_id INT,
    rating INT NOT NULL,
    review_text TEXT,
    review_date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (product_id) REFERENCES products(product_id),
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
);
```

## ER Diagram




## Entity Relationship

| Parent Table | Child Table | Relationship Type | Foreign Key | Real-Life Meaning |
|---|---|---|---|---|
| customers | orders | One-to-Many | orders.customer_id → customers.customer_id | A customer can place multiple orders |
| orders | order_items | One-to-Many | order_items.order_id → orders.order_id | An order can contain multiple items |
| products | order_items | One-to-Many | order_items.product_id → products.product_id | A product can appear in many different orders |
| orders | payments | One-to-Many | payments.order_id → orders.order_id | One or more than one payment per order. |
| products | product_reviews | One-to-Many | product_reviews.product_id → products.product_id | A product can have many customer reviews |
| customers | product_reviews | One-to-Many | product_reviews.customer_id → customers.customer_id | A customer can write multiple reviews |

# Data Query Language

**Level 1: Basics**

1. SELECT with column names
2. SELECT * (all columns)
3. DISTINCT keyword
4. WHERE clause with:
    - Comparison operators (=, !=, <, >, <=, >=)
    - BETWEEN, IN, NOT IN, LIKE
5. Logical operators: AND, OR, NOT
6. Sorting: ORDER BY (ASC/DESC)
7. LIMIT / TOP (based on RDBMS)

---

**Level 2: Filtering and Formatting**

8. Using IS NULL / IS NOT NULL
9. Aliases: AS for columns and tables
10. Simple arithmetic operations in SELECT
11. Column concatenation (e.g., first name + last name)
12. Formatting date/time columns (e.g., DATE(order_date))

---

**Level 3: Aggregations**

13. Aggregate functions:

- COUNT(), SUM(), AVG(), MIN(), MAX()

14. GROUP BY clause
15. HAVING vs WHERE
16. GROUP BY with multiple columns

**Level 4: Multi-Table Queries (JOINS)**

17. INNER JOIN
18. LEFT JOIN
19. RIGHT JOIN (optional for MySQL learners)
20. FULL OUTER JOIN (if supported by DB)
21. SELF JOIN (e.g., employee hierarchy)
22. Multiple joins in one query

**Level 5: Subqueries (Inner Queries)**

23. Scalar subquery in SELECT
24. Subquery in WHERE (e.g., WHERE price > (SELECT AVG(price) ...))
25. Subquery in FROM (inline views)
26. EXISTS / NOT EXISTS usage

**Level 6: Set Operations**

27. UNION / UNION ALL
28. INTERSECT, EXCEPT (if supported)

# Questions

## Level 1: Basics

**1. Retrieve customer names and emails for email marketing**
This helps the marketing team extract basic customer contact details for campaigns.

**2. View complete product catalog with all available details**
The product manager may want to review all product listings in one go.

**3. List all unique product categories**
Useful for analyzing the range of departments or for creating filters on the website.

**4. Show all products priced above ₹1,000**
This helps identify high-value items for premium promotions or pricing strategy reviews.

**5. Display products within a mid-range price bracket (₹2,000 to ₹5,000)**
A merchandising team might need this to create a mid-tier pricing campaign.

**6. Fetch data for specific customer IDs (e.g., from loyalty program list)**
This is used when customer IDs are pre-selected from another system.

**7. Identify customers whose names start with the letter 'A'**
Used for alphabetical segmentation in outreach or app display.

**8. List electronics products priced under ₹3,000**
Used by merchandising or frontend teams to showcase budget electronics.

**9. Display product names and prices in descending order of price**
This helps teams easily view and compare top-priced items.

**10. Display product names and prices, sorted by price and then by name**
The merchandising or catalog team may want to list products from most expensive to cheapest. If multiple products have the same price, they should be sorted alphabetically for clarity on storefronts or printed catalogs.

## Level 2: Filtering and Formatting

**1. Retrieve orders where customer information is missing (possibly due to data migration or deletion)**
Used to identify orphaned orders or test data where customer_id is not linked.

**2. Display customer names and emails using column aliases for frontend readability**
Useful for feeding into frontend displays or report headings that require user-friendly labels.

**3. Calculate total value per item ordered by multiplying quantity and item price**
This can help generate per-line item bill details or invoice breakdowns.

**4. Combine customer name and phone number in a single column**
Used to show brief customer summaries or contact lists.

**5. Extract only the date part from order timestamps for date-wise reporting**
Helps group or filter orders by date without considering time.

**6. List products that do not have any stock left**
This helps the inventory team identify out-of-stock items.

## Level 3: Aggregations

**1. Count the total number of orders placed**
Used by business managers to track order volume over time.

**2. Calculate the total revenue collected from all orders**
This gives the overall sales value.

**3. Calculate the average order value**
Used for understanding customer spending patterns.

**4. Count the number of customers who have placed at least one order**
This identifies active customers.

**5. Find the number of orders placed by each customer**
Helpful for identifying top or repeat customers.

**6. Find total sales amount made by each customer**

**7. List the number of products sold per category**
This helps category managers assess performance by department.

**8. Find the average item price per category**
Useful to compare pricing across departments.

**9. Show number of orders placed per day**
Used to track daily business activity and demand trends.

**10. List total payments received per payment method**
Helps the finance team understand preferred transaction modes.

## Level 4: Multi-Table Queries (JOINS)

**1. Retrieve order details along with the customer name (INNER JOIN)**
Used for displaying which customer placed each order.

**2. Get list of products that have been sold (INNER JOIN with order_items)**
Used to find which products were actually included in orders.

**3. List all orders with their payment method (INNER JOIN)**

Used by finance or audit teams to see how each order was paid for.

**4. Get list of customers and their orders (LEFT JOIN)**
Used to find all customers and see who has or hasn't placed orders.

**5. List all products along with order item quantity (LEFT JOIN)**
Useful for inventory teams to track what sold and what hasn't.

**6. List all payments including those with no matching orders (RIGHT JOIN)**
Rare but used when ensuring all payments are mapped correctly.

**7. Combine data from three tables: customer, order, and payment**
Used for detailed transaction reports.

## Level 5: Subqueries (Inner Queries)

**1. List all products priced above the average product price**
Used by pricing analysts to identify premium-priced products.

**2. Find customers who have placed at least one order**
Used to identify active customers for loyalty campaigns.

**3. Show orders whose total amount is above the average for that customer**
Used to detect unusually high purchases per customer.

**4. Display customers who haven't placed any orders**
Used for re-engagement campaigns targeting inactive users.

**5. Show products that were never ordered**
Helps with inventory clearance decisions or product deactivation.

**6. Show highest value order per customer**
Used to identify the largest transaction made by each customer.

**7. Highest Order Per Customer (Including Names)**
Used to identify the largest transaction made by each customer. Outputs name as well.

## Level 6: Set Operations

**1. List all customers who have either placed an order or written a product review**
Used to identify engaged customers from different activity areas.

**2. List all customers who have placed an order as well as reviewed a product [intersect not supported]**
Used to identify highly engaged customers for rewards.