## E-Commerce Database – Technical Report

### 1) Purpose and Scope

This project aims to design a **relational e-commerce database**, populate the tables with sample data, and validate the relationships between tables using various **SELECT** and **JOIN** queries.
Additionally, the table structures were tested with sample data, and relationships were verified through JOIN queries.

### 2) Database Schema and Tables

The database consists of **six main tables**, each designed to store different types of data:

- **Customers:** Stores customer details (first name, last name, email, city). customer_id is defined as the **primary key (PK)**.
- **Orders:** Stores order information. Each order links to the **Customers** table through the customer_id field as a **foreign key (FK)**.
- **Products:** Stores product names and pricing information. Also connects to the **Categories** table via category_id.
- **Categories:** Defines product categories. category_id is the **primary key (PK)**.
- **Payments:** Stores payment details for each order. The order_id field links to the **Orders** table with a **UNIQUE** constraint, ensuring a **1:1 relationship**.
- **OrderItems:** Serves as a bridge table that stores detailed information about products within orders, establishing an **M:N relationship** between **Orders** and **Products**.

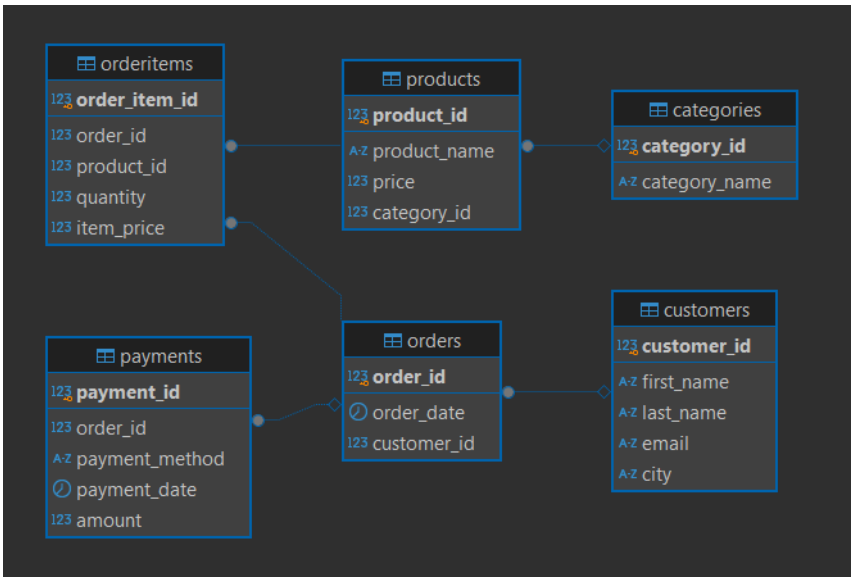
**Note:** The schema clearly reflects **1:N** and **1:1** relationships using PK and FK definitions.

### 3) Relationships Between Tables

The designed database establishes the following relationships:

- **Customers (1) → (N) Orders**
  One customer can place multiple orders.
- **Orders (1) → (1) Payments**
  Each order has exactly one payment.
- **Categories (1) → (N) Products**
  Each category can include multiple products.
- **Orders (1) → (N) OrderItems** and **Products (1) → (N) OrderItems**
  Orders and products are connected through the **OrderItems** table, forming an **M:N relationship**.

These relationships are visually represented in the **ER diagram** prepared for the project.




**4) Sample Data**

To test the database structure, **INSERT** statements were used to add sample data into all tables. Sample entries for products, customers, orders, payments, and order items were added to **test the system end-to-end**.

**5) Validation Queries**

Several queries were created to ensure the database works correctly:

- **Basic checks:**
  SELECT * queries were used to inspect table contents.
- **JOIN tests:**
  - **Customer ↔ Orders:** Combined customer and order data to list orders per customer.
  - **Products ↔ Categories:** Joined product and category data to identify which product belongs to which category.
  - **Orders ↔ Payments:** Merged orders and payment data to validate payment details.
- Additionally, by joining the **OrderItems** table with **Orders** and **Products**, the correctness of the **many-to-many relationship** was verified.

**6) Compliance with Evaluation Criteria**

- **Database design:** Primary and foreign keys were properly defined, ensuring accurate table relationships.
- **Data types:** VARCHAR, DATE, and DECIMAL were used appropriately according to the scenario.

- **Sample data:** Example records were inserted into all tables to test the system end-to-end.
- **Queries:** SELECT and JOIN queries were successfully used to validate table relationships.
- **Normalization:**
  The database complies with **1NF** and **2NF** rules:
  - Each column stores **atomic data**, which ensures **1NF**.
  - Since all primary keys consist of a **single column**, there is **no partial dependency**, meaning **2NF** is also satisfied.
  - Additionally, in the **OrderItems** table, the natural candidate key (order_id, product_id) fully determines the dependent attributes quantity and item_price.

**7) Recommendations for Improvement**

While the project meets the basic requirements, the following enhancements could improve performance and data quality:

- **Data integrity:**
  Add a **UNIQUE** and **NOT NULL** constraint for the email field in the **Customers** table.
- **Validation constraints:**
  Apply **CHECK (>=0)** constraints to numeric fields like price, amount, and item_price.
- **Cascade deletions:**
  Use **ON DELETE CASCADE** on foreign keys to automatically remove dependent records.
- **Performance optimization:**
  Add **indexes** on frequently used FK fields such as orders.customer_id and orderitems.order_id for faster queries.