E-Commerce Database – Technical Report

1) Purpose and Scope

This project aims to design a **relational database** for an e-commerce scenario, 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 the relationships were verified through SELECT and JOIN operations.

2) Database Schema and Tables

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

- **Customers:** Stores customer information (first name, last name, email, city). The customer_id field is defined as the **primary key (PK)**.
- **Orders:** Stores order information. Each order is linked to the **Customers** table via the customer_id field as a **foreign key (FK)**.
- **Products:** Stores product names and pricing information. Each product is linked to the **Categories** table via the category_id field.
- **Categories:** Defines product categories, with category_id used as the **primary key (PK)**.
- **Payments:** Contains payment details for each order. The order_id field links to the **Orders** table, and a **UNIQUE constraint** ensures a **1:1 relationship** between an order and its payment.
- **OrderItems:** Serves as a bridge table to store details of products included in each order. It establishes a **many-to-many (M:N) relationship** between **Orders** and **Products**.

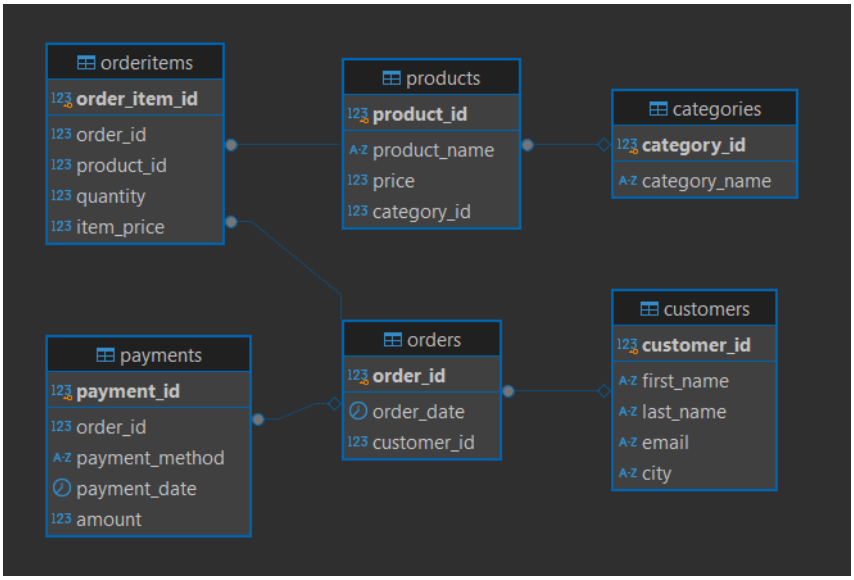
Note: The schema clearly reflects **1:N** and **1:1** relationships through the definition of **primary** and **foreign keys**.

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**
  One category can include multiple products.
- **Orders (1) → (N) OrderItems** and **Products (1) → (N) OrderItems**
  Orders and products are connected via **OrderItems**, establishing an **M:N relationship**.

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



4) Sample Data

To test the database structure, **sample data** was inserted into all tables using **INSERT** statements. Products, customers, orders, payments, and order items were populated to perform **end-to-end testing** of the system.

5) Validation Queries

To ensure the database was functioning correctly, several queries were prepared and executed:

- **Basic checks:**
  Verified table contents using SELECT * queries.
- **JOIN tests:**
  - **Customer ↔ Orders:** Joined customer and order data to list orders by customer.
  - **Products ↔ Categories:** Joined products and categories to display which products belong to which category.
  - **Orders ↔ Payments:** Joined orders and payment details to validate 1:1 payment relationships.
- Additionally, the **OrderItems** table was used to join **Orders** and **Products** to confirm the **many-to-many** relationship works correctly.

6) Steps to Run the Project

1. **Create a new database** or schema *(optional)*.
2. Run the ecommerce_project.sql script using **pgAdmin** or **DBeaver**.

   The script includes steps for **table creation**, **data insertion**, and **validation queries**.

3. Use the database tool to **automatically generate the ER diagram** and visually confirm the table relationships.

7) Compliance with Evaluation Criteria

- **Database Design:** Primary keys and foreign keys were correctly defined to establish proper relationships between tables.
- **Data Types:** Appropriate data types were chosen based on the scenario, such as VARCHAR, DATE, and DECIMAL.
- **Sample Data:** Sample 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.

8) Recommendations for Improvement

While the project meets basic requirements, the following enhancements could be implemented:

- **Data Integrity:**
  Add a **UNIQUE** and **NOT NULL** constraint for the email field in the **Customers** table to ensure email uniqueness.
- **Validation Constraints:**
  Apply **CHECK** constraints to numeric fields such as price, amount, and item_price to ensure values are greater than or equal to zero.
- **Data Deletion Scenarios:**
  Use **ON DELETE CASCADE** for foreign keys to automatically remove dependent records when a parent record is deleted.
- **Performance Optimization:**
  Create **indexes** on frequently used foreign key fields such as orders.customer_id and orderitems.order_id to improve query performance.