E-COMMERCE DATABASE SYSTEM PROJECT

Cover Page

Project Title: E-Commerce Database Management System

Student Name / Number: Muhammet Burak KILIÇ / 212020050019

Course Name: MIS-3016 Databases

Instructor: DR. Mesut Ünlü

Submission Date: 19.06.2025

Table of Contents

Introduction

Domain Introduction

E-commerce, also known as electronic commerce, encompasses all commercial activities involving the buying and selling of products and services over the internet. With the acceleration of digital transformation today, the e-commerce sector has become a continuously growing and developing field. The e-commerce domain selected for this project includes a comprehensive system structure where customers can purchase products online, place orders, and process payments through online platforms.

Project Purpose and Scope

The primary purpose of this project is to design the database structure of a modern e-commerce system and implement this system using SQL. The project scope focuses on:

- Customer information management
- Product catalog and category system creation
- Order management system establishment
- Payment transaction tracking
- Address information storage

The system has been designed to meet the basic functions of real-world e-commerce platforms and uses a normalized database structure.

Database Design

Conceptual Model

Entity-Relationship Diagram

Our system contains 7 main entities:

Entities:

1. **USERS** - Stores system user information
2. **ADDRESSES** - Contains user address information
3. **CATEGORIES** - Defines product categories
4. **PRODUCTS** - Stores information about products for sale
5. **ORDERS** - Contains main information about placed orders
6. **ORDER_DETAILS** - Stores detailed information about order contents
7. **PAYMENTS** - Contains payment transaction information

Relationships:

- Users → Addresses (1:N) - A user can have multiple addresses
- Users → Orders (1:N) - A user can place multiple orders
- Categories → Products (1:N) - A category can contain multiple products
- Orders → Order_Details (1:N) - An order can contain multiple products
- Products → Order_Details (1:N) - A product can be in multiple orders
- Orders → Payments (1:1) - Each order has one payment record
- Addresses → Orders (1:N) - Multiple orders can be delivered to the same address

Logical Model

Normalization Explanations

Our system has been designed in accordance with 3NF (Third Normal Form) rules:

1NF (First Normal Form):

- Atomic values are used in all tables
- Repeating groups have been eliminated
- Each row can be uniquely identified

2NF (Second Normal Form):

- All non-key attributes are fully dependent on the primary key
- Partial functional dependencies have been eliminated

3NF (Third Normal Form):

- Transitive dependencies have been eliminated
- Category information in the products table has been normalized by adding the Categories table

Table Structures and Relationships

PRIMARY KEYS:

- An automatically incrementing id field of SERIAL type is used in each table

FOREIGN KEYS:

- addresses.user_id → users.id
- products.category_id → categories.id
- orders.user_id → users.id
- orders.address_id → addresses.id
- order_details.order_id → orders.id
- order_details.product_id → products.id
- payments.order_id → orders.id

Physical Model

Data Types and Constraints

Data Types:

- SERIAL: For automatically incrementing numeric values
- VARCHAR(n): For variable-length text fields
- TEXT: For long text content
- DECIMAL(10,2): For currency values
- INT: For integer values
- TIMESTAMP: For date and time information

Constraints:

- NOT NULL: For mandatory fields
- UNIQUE: For unique values (email)
- CHECK: To control specific values (payment_type, status)
- ON DELETE CASCADE: For deleting related records
- ON DELETE SET NULL: For assigning null values to related records

---

SQL Implementation

Database and Table Creation

Users Table

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password TEXT NOT NULL,
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

This table stores basic information about system users. The email field is defined as unique (UNIQUE) and the registration date is automatically assigned.

Addresses Table

```sql
CREATE TABLE addresses (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    address_title VARCHAR(50),
    city VARCHAR(50),
    district VARCHAR(50),
    neighborhood VARCHAR(50),
    street VARCHAR(50),
    postal_code VARCHAR(10)
);
```

Stores user address information. When a user is deleted (ON DELETE CASCADE), all addresses belonging to that user are also deleted.

Categories Table

```sql
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    description TEXT
);
```

Stores product categories and has been separated from the products table for normalization.

Products Table

```sql
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10,2) NOT NULL,
    stock INT NOT NULL,
    category_id INT REFERENCES categories(id) ON DELETE SET NULL
);
```

Stores product information. When a category is deleted (ON DELETE SET NULL), the product's category reference becomes null.

Orders Table

```sql
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    address_id INT REFERENCES addresses(id) ON DELETE SET NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Stores main order information and the order date is automatically assigned.

Order_Details Table

```sql
CREATE TABLE order_details (
    id SERIAL PRIMARY KEY,
    order_id INT REFERENCES orders(id) ON DELETE CASCADE,
    product_id INT REFERENCES products(id) ON DELETE CASCADE,
    quantity INT NOT NULL,
    unit_price DECIMAL(10,2) NOT NULL
);
```

Stores order details and establishes a many-to-many relationship between orders and products tables.

Payments Table

```sql
CREATE TABLE payments (
    id SERIAL PRIMARY KEY,
    order_id INT REFERENCES orders(id) ON DELETE CASCADE,
    payment_type VARCHAR(20) CHECK (payment_type IN ('Credit Card', 'Bank Transfer', 'Cash on D
    status VARCHAR(20) CHECK (status IN ('Pending', 'Approved', 'Cancelled')),
    date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Stores payment information and CHECK constraints ensure specific values are entered.

Sample Data Insertion

User Data

```sql
INSERT INTO users (first_name, last_name, email, password) VALUES
('John', 'Doe', 'john.doe@example.com', 'hashed_password_1'),
('Jane', 'Smith', 'jane.smith@example.com', 'hashed_password_2'),
('Michael', 'Johnson', 'michael.johnson@example.com', 'hashed_password_3'),
('Emily', 'Wilson', 'emily.wilson@example.com', 'hashed_password_4'),
('David', 'Brown', 'david.brown@example.com', 'hashed_password_5');
```

Address Data

```sql
INSERT INTO addresses (user_id, address_title, city, district, neighborhood, street, postal_cod
(1, 'Home', 'New York', 'Manhattan', 'Midtown', '5th Avenue', '10001'),
(1, 'Work', 'New York', 'Brooklyn', 'Williamsburg', 'Bedford Ave', '11211'),
(2, 'Home', 'Los Angeles', 'Downtown', 'Arts District', 'Spring St', '90013'),
(3, 'Home', 'Chicago', 'Lincoln Park', 'Lakeview', 'Clark St', '60614'),
(4, 'Home', 'Miami', 'Miami Beach', 'South Beach', 'Ocean Drive', '33139'),
(5, 'Home', 'Seattle', 'Capitol Hill', 'Broadway', 'Pike St', '98122');
```

Category Data

sql

```sql
INSERT INTO categories (name, description) VALUES
('Electronics', 'Electronic devices and accessories'),
('Clothing', 'Apparel and fashion items'),
('Books', 'Books, e-books, and publications'),
('Home & Kitchen', 'Household items and kitchen appliances'),
('Sports & Outdoors', 'Sports equipment and outdoor gear');
```

Product Data

sql

```sql
INSERT INTO products (name, description, price, stock, category_id) VALUES
('Smartphone X', 'Latest model with high-resolution camera', 799.99, 50, 1),
('Laptop Pro', '15-inch laptop with SSD', 1299.99, 30, 1),
('Wireless Headphones', 'Noise-cancelling Bluetooth headphones', 149.99, 100, 1),
('T-shirt Basic', 'Cotton t-shirt in various colors', 19.99, 200, 2),
('Jeans Classic', 'Denim jeans with straight fit', 59.99, 75, 2),
('Novel Bestseller', 'Fiction bestseller of the year', 24.99, 120, 3),
('Programming Guide', 'Comprehensive guide to modern programming', 39.99, 45, 3),
('Coffee Maker', 'Automatic coffee maker with timer', 89.99, 25, 4),
('Cooking Pot Set', '5-piece non-stick cooking pot set', 129.99, 15, 4),
('Running Shoes', 'Lightweight running shoes for all terrains', 79.99, 60, 5);
```

Basic Queries

List All Users

sql

```sql
SELECT * FROM users;
```

List All Categories

sql

```sql
SELECT * FROM categories;
```

Products with Low Stock

```sql
SELECT id, name, stock, price
FROM products
WHERE stock < 30
ORDER BY stock ASC;
```

This query lists products with stock quantity less than 30 in ascending order by stock quantity.

Products Listed by Categories

```sql
SELECT p.id, p.name, p.description, p.price, p.stock, c.name AS category_name
FROM products p
LEFT JOIN categories c ON p.category_id = c.id
ORDER BY p.id;
```

Queries with Related Tables

User Total Order Analysis

```sql
SELECT u.id, u.first_name, u.last_name, COUNT(o.id) AS total_orders,
       SUM(o.total_amount) AS total_spent
FROM users u
LEFT JOIN orders o ON u.id = o.user_id
GROUP BY u.id, u.first_name, u.last_name
ORDER BY total_spent DESC;
```

This query shows each user's total number of orders and spending amount.

Best Selling Products

```sql
SELECT p.id, p.name, SUM(od.quantity) AS total_quantity_ordered
FROM products p
JOIN order_details od ON p.id = od.product_id
GROUP BY p.id, p.name
ORDER BY total_quantity_ordered DESC
LIMIT 5;
```

Sales Analysis by Categories

sql

```sql
SELECT c.name AS category_name,
       SUM(od.quantity * od.unit_price) AS total_sales,
       COUNT(DISTINCT od.order_id) AS number_of_orders
FROM categories c
JOIN products p ON c.id = p.category_id
JOIN order_details od ON p.id = od.product_id
GROUP BY c.name
ORDER BY total_sales DESC;
```

Comprehensive Order Report

sql

```sql
SELECT o.id AS order_id,
       o.order_date,
       CONCAT(u.first_name, ' ', u.last_name) AS customer_name,
       p.name AS product_name,
       od.quantity,
       od.unit_price,
       (od.quantity * od.unit_price) AS subtotal,
       o.total_amount AS order_total,
       pm.payment_type,
       pm.status AS payment_status
FROM orders o
JOIN users u ON o.user_id = u.id
JOIN order_details od ON o.id = od.order_id
JOIN products p ON od.product_id = p.id
JOIN payments pm ON o.id = pm.order_id
ORDER BY o.order_date DESC;
```

Users with Home Addresses

sql

```sql
SELECT u.id, u.first_name, u.last_name, u.email,
       a.city, a.district, a.neighborhood, a.street
FROM users u
JOIN addresses a ON u.id = a.user_id
WHERE a.address_title = 'Home';
```

Payment Type Analysis

sql

```sql
SELECT payment_type,
       COUNT(*) AS number_of_payments,
       SUM(o.total_amount) AS total_amount
FROM payments p
JOIN orders o ON p.order_id = o.id
GROUP BY payment_type
ORDER BY number_of_payments DESC;
```

User Orders with Payment Status

sql

```sql
SELECT o.id AS order_id, o.order_date, o.total_amount, p.status AS payment_status
FROM orders o
JOIN payments p ON o.id = p.order_id
WHERE o.user_id = 1
ORDER BY o.order_date DESC;
```

Frequently Bought Together Products

sql

```sql
SELECT p.id, p.name, p.price, COUNT(*) AS frequency
FROM products p
JOIN order_details od ON p.id = od.product_id
WHERE od.order_id IN (
    SELECT DISTINCT o.id
    FROM orders o
    JOIN order_details od ON o.id = od.order_id
    WHERE od.product_id = 1
      AND o.id != 1
)
AND p.id != 1
GROUP BY p.id, p.name, p.price
ORDER BY frequency DESC
LIMIT 5;
```

Electronics Products Over $500

```sql
SELECT p.name, p.price, c.name AS category_name
FROM products p
JOIN categories c ON p.category_id = c.id
WHERE c.name = 'Electronics' AND p.price > 500;
```

Data Update and Deletion

Product Price Update

```sql
UPDATE products
SET price = 849.99
WHERE id = 1;
```

Add View Count Column

```sql
ALTER TABLE products ADD COLUMN view_count INT DEFAULT 0;
```

Stock Update

```sql
UPDATE products
SET stock = stock - 1
WHERE id = 1;
```

This query decreases the stock quantity of a specific product by one (post-sale stock update).

Address Deletion

```sql
DELETE FROM addresses WHERE id = 2;
```

Views

Product Detail View

```sql
CREATE VIEW product_details_with_category AS
SELECT
    p.id AS product_id,
    p.name AS product_name,
    p.description AS product_description,
    p.price,
    p.stock,
    c.name AS category_name,
    c.description AS category_description
FROM
    products p
LEFT JOIN
    categories c ON p.category_id = c.id;
```

This view combines product information with category information, defining a frequently used query as a view.

Using the View

```sql
SELECT * FROM product_details_with_category;
```

Dropping the View

```sql
DROP VIEW IF EXISTS product_details_with_category;
```

Conclusion

Overall Project Evaluation

This e-commerce database project has been successfully designed and implemented to meet the basic requirements of a modern online sales platform. Within the project scope:

- A normalized database structure has been created
- A comprehensive system has been established with 7 main tables
- Referential integrity has been ensured
- Various SQL commands and queries have been successfully implemented
- Sample data suitable for real-life scenarios has been used

The system supports all basic e-commerce processes from user management to product catalog, from order tracking to payment processing.

Challenges Encountered and Recommendations

Challenges Encountered:

- Performance optimization of multi-table JOIN operations
- Determining appropriate constraints to ensure data integrity
- Eliminating data redundancy during the normalization process

Recommendations:

- Creating appropriate indexes for large datasets
- Using stored procedures for performance improvement
- Implementing data backup and security measures
- Creating automatic business processes using triggers

Future system enhancements could include product reviews, favorites, discount coupons, and more detailed inventory management.

References

1. Date, C.J. (2003). *An Introduction to Database Systems*. 8th Edition, Addison-Wesley.
2. Elmasri, R., & Navathe, S.B. (2010). *Fundamentals of Database Systems*. 6th Edition, Pearson.
3. PostgreSQL Documentation. (2024). *PostgreSQL 16 Documentation*. https://www.postgresql.org/docs/
4. Silberschatz, A., Galvin, P.B., & Gagne, G. (2018). *Database System Concepts*. 7th Edition, McGraw-Hill.
5. W3Schools SQL Tutorial. (2024). *SQL Tutorial*. https://www.w3schools.com/sql/

Appendices

Appendix A: Table Structure Summary

Table	Primary Key	Foreign Keys	Record Count
users	id	-	5
addresses	id	user_id	6
categories	id	-	5
products	id	category_id	10
orders	id	user_id, address_id	6
order_details	id	order_id, product_id	11
payments	id	order_id	6

Appendix B: Relationship Matrix

Relationships established in this system:

- Users ↔ Addresses (1:N)
- Users ↔ Orders (1:N)
- Categories ↔ Products (1:N)
- Orders ↔ Order_Details (1:N)
- Products ↔ Order_Details (1:N)
- Orders ↔ Payments (1:1)
- Addresses ↔ Orders (1:N)

Appendix C: Performance Recommendations

Recommended indexes to improve system performance:

```sql
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_products_category ON products(category_id);
CREATE INDEX idx_orders_user ON orders(user_id);
CREATE INDEX idx_order_details_order ON order_details(order_id);
CREATE INDEX idx_order_details_product ON order_details(product_id);
```