Introduction

This documentation outlines the design of a database system for an e-commerce platform. The system includes various modules such as product listings, user accounts, shopping carts, orders, payments, and inventory management. The goal is to provide a comprehensive and scalable database schema that ensures efficient data management and integrity.

Requirements

Functional Requirements

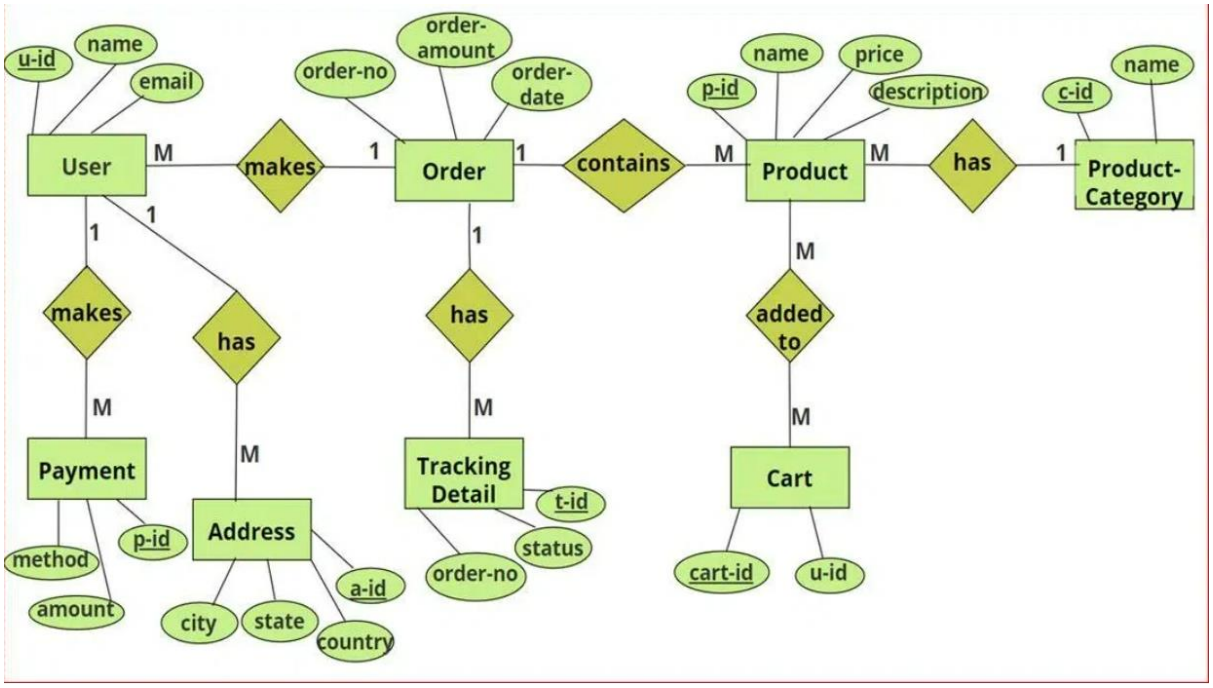
1. **User Accounts**: Manage user registration, login, and profile information.
2. **Product Listings**: Maintain product details, categories, and pricing information.
3. **Shopping Carts**: Handle the addition, removal, and modification of items in a user's shopping cart.
4. **Orders**: Manage the creation and tracking of user orders, including order items and status.
5. **Payments**: Process and record payment information associated with orders.
6. **Inventory Management**: Track product inventory levels and update them based on sales.

Non-Functional Requirements

1. **Scalability**: The database should handle increasing amounts of data and concurrent transactions.
2. **Security**: Ensure user data, especially payment information, is stored securely.
3. **Integrity**: Maintain data integrity through the use of foreign keys and constraints.
4. **Performance**: Optimize queries for fast data retrieval and updates.

ER Diagram

The Entity-Relationship (ER) diagram visually represents the database structure and relationships between different entities.



Implementation

Users Table

Stores information about the users of the platform.

```
CREATE TABLE Users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    address TEXT,
    phone_number VARCHAR(15),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Products Table

Stores information about products available on the platform.

```
CREATE TABLE Products (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    category_id INT,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL,
    image_url VARCHAR(255),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES Categories(category_id)
);
```

Categories Table

Stores product categories.

```
CREATE TABLE Categories (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL,
    description TEXT
);
```

Shopping Carts Table

Stores shopping cart items for each user.

```
CREATE TABLE ShoppingCarts (
    cart_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    product_id INT,
    quantity INT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES Users(user_id),

    FOREIGN KEY (product_id) REFERENCES Products(product_id)

);
```

Orders Table

Stores orders placed by users.

```
CREATE TABLE Orders (

    order_id INT AUTO_INCREMENT PRIMARY KEY,

    user_id INT,

    total_amount DECIMAL(10, 2) NOT NULL,

    status ENUM('pending', 'completed', 'cancelled') DEFAULT 'pending',

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    FOREIGN KEY (user_id) REFERENCES Users(user_id)

);
```

Order Items Table

Stores individual items within an order.

```
CREATE TABLE OrderItems (

    order_item_id INT AUTO_INCREMENT PRIMARY KEY,

    order_id INT,

    product_id INT,

    quantity INT NOT NULL,

    price DECIMAL(10, 2) NOT NULL,

    total DECIMAL(10, 2) NOT NULL,

    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
```

```
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

Payments Table

Stores payment information for orders.

```
CREATE TABLE Payments (
    payment_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    amount DECIMAL(10, 2) NOT NULL,
    payment_method ENUM('credit_card', 'paypal', 'bank_transfer') NOT NULL,
    payment_status ENUM('pending', 'completed', 'failed') DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
```

Inventory Table

Stores inventory details for products.

```
CREATE TABLE Inventory (
    inventory_id INT AUTO_INCREMENT PRIMARY KEY,
    product_id INT,
    quantity INT NOT NULL,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

Relationships

1. **Users and ShoppingCarts**: A user can have multiple shopping cart items. Each shopping cart item is linked to a user and a product.
    - `Users` (1) ---< (M) `ShoppingCarts`
    - `Products` (1) ---< (M) `ShoppingCarts`

2. **Users and Orders**: A user can place multiple orders. Each order belongs to one user.
    - `Users` (1) ---< (M) `Orders`
3. **Orders and OrderItems**: An order can contain multiple items. Each order item is linked to an order and a product.
    - `Orders` (1) ---< (M) `OrderItems`
    - `Products` (1) ---< (M) `OrderItems`
4. **Orders and Payments**: Each order can have multiple associated payments, although typically, one order corresponds to one payment transaction.
    - `Orders` (1) ---< (M) `Payments`
5. **Products and Inventory**: Each product has an associated inventory record to manage stock levels.
    - `Products` (1) ---< (1) `Inventory`
6. **Products and Categories**: Each product belongs to one category, and each category can have multiple products.
    - `Categories` (1) ---< (M) `Products`

Sample SQL Queries

1. Retrieve All Products in a Category

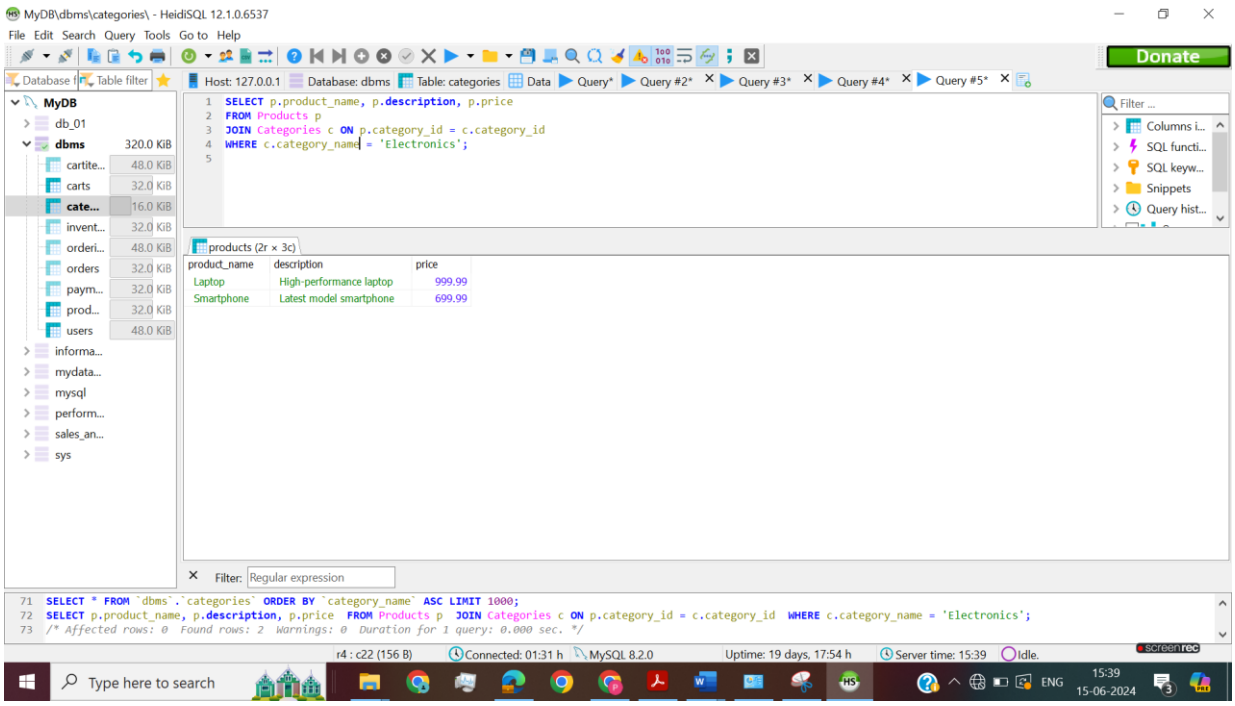
SELECT p.name, p.description, p.price

FROM Products p

JOIN Categories c ON p.category_id = c.category_id

WHERE c.name = 'Electronics';

Output:



2. Get User's Shopping Cart Items

```
SELECT p.product_name, sc.quantity, p.price, (sc.quantity * p.price) AS
total_price
FROM cartitems sc
JOIN Products p ON sc.product_id = p.product_id
WHERE sc.product_id = 1;
```

Output: