Answer The Question

Question A: Design a relational database to store all the information contained in the above images such as products, addresses, stores, categories, orders, users, ... And describe the types of database normalization you used.

- This’s erd I draw with lucidchart. You need login to google screen:

https://lucid.app/lucidchart/1e3e42ec-7335-4792-8d23-12be17012b51/edit?invitationId=inv_2e0e27a6-96f1-405b-9ad8-54b25926a466&page=0_0#

```
DROP DATABASE IF EXISTS ecommerce;
CREATE DATABASE ecommerce;
USE ecommerce;

CREATE TABLE users (
                user_id INT PRIMARY KEY AUTO_INCREMENT,
                name VARCHAR(100) NOT NULL,
                email VARCHAR(100) NOT NULL UNIQUE,
```

```
                phone VARCHAR(20) UNIQUE,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UI
);

CREATE TABLE provinces (
                province_id INT PRIMARY KEY,
                name VARCHAR(100) NOT NULL
);

CREATE TABLE districts (
                district_id INT PRIMARY KEY,
                name VARCHAR(100) NOT NULL,
                province_id INT NOT NULL,
                FOREIGN KEY (province_id) REFERENCES provinces(province
);

CREATE TABLE communes (
                commune_id INT PRIMARY KEY,
                name VARCHAR(100) NOT NULL,
                district_id INT NOT NULL,
                FOREIGN KEY (district_id) REFERENCES districts(district_id)
);

CREATE TABLE housing_types (
                    housing_type_id INT PRIMARY KEY,
                    name VARCHAR(50) NOT NULL
);

CREATE TABLE addresses (
                address_id INT PRIMARY KEY AUTO_INCREMENT,
                user_id INT NOT NULL,
                commune_id INT NOT NULL,
                street VARCHAR(100),
                housing_type_id INT,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
                FOREIGN KEY (user_id) REFERENCES users(user_id),
```

```
                    FOREIGN KEY (commune_id) REFERENCES communes(comm
                    FOREIGN KEY (housing_type_id) REFERENCES housing_types
);

CREATE TABLE categories (
                     category_id INT PRIMARY KEY AUTO_INCREMENT,
                     name VARCHAR(100) NOT NULL
);

CREATE TABLE brands (
                   brand_id INT PRIMARY KEY AUTO_INCREMENT,
                   name VARCHAR(100) NOT NULL
);

CREATE TABLE category_discounts (
                         category_discount_id INT PRIMARY KEY AUTO_INCREME
                         category_id INT NOT NULL,
                         discount_percent DECIMAL(5,2),
                         valid_from DATE,
                         valid_to DATE,
                         FOREIGN KEY (category_id) REFERENCES categories(ca
);

CREATE TABLE products (
                    product_id INT PRIMARY KEY AUTO_INCREMENT,
                    name VARCHAR(100) NOT NULL,
                    category_id INT NOT NULL,
                    brand_id INT NOT NULL,
                    base_price DECIMAL(10,2),
                    description TEXT,
                    gender VARCHAR(20) NOT NULL,
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
                    FOREIGN KEY (category_id) REFERENCES categories(category
                    FOREIGN KEY (brand_id) REFERENCES brands(brand_id)
);

CREATE TABLE product_discounts (
```

```
                    product_discount_id INT PRIMARY KEY AUTO_INCREMEN
                    product_id INT NOT NULL,
                    discount_percent DECIMAL(5,2),
                    valid_from DATE,
                    valid_to DATE,
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                    FOREIGN KEY (product_id) REFERENCES products(produ
);

CREATE TABLE product_variants (
                    variant_id INT PRIMARY KEY AUTO_INCREMENT,
                    product_id INT NOT NULL,
                    color VARCHAR(30),
                    size VARCHAR(10),
                    stock_quantity INT,
                    price_adjustment DECIMAL(12,2),
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                    FOREIGN KEY (product_id) REFERENCES products(produc
);

CREATE TABLE stores (
                store_id INT PRIMARY KEY AUTO_INCREMENT,
                name VARCHAR(100) NOT NULL,
                location VARCHAR(255),
                description VARCHAR(255),
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON U
);

CREATE TABLE product_stocks (
                    id SERIAL PRIMARY KEY,
                    product_id INT NOT NULL,
                    store_id INT NOT NULL,
                    quantity_on_hand INT DEFAULT 0,
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP (
```

```
                    FOREIGN KEY (product_id) REFERENCES products(product
                    FOREIGN KEY (store_id) REFERENCES stores(store_id),
                    UNIQUE (product_id, store_id)
);

CREATE TABLE vouchers (
                voucher_id INT PRIMARY KEY AUTO_INCREMENT,
                code VARCHAR(50) UNIQUE,
                discount_percent DECIMAL(5,2),
                valid_from DATE,
                valid_to DATE,
                voucher_type VARCHAR(255) NOT NULL,
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
);

CREATE TABLE orders (
                order_id INT PRIMARY KEY AUTO_INCREMENT,
                user_id INT NOT NULL,
                address_id INT NOT NULL,
                order_date DATETIME,
                payment_method VARCHAR(30),
                total_amount DECIMAL(12,2),
                created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON U
                FOREIGN KEY (user_id) REFERENCES users(user_id),
                FOREIGN KEY (address_id) REFERENCES addresses(address_id
);

CREATE TABLE order_details (
                    order_detail_id INT PRIMARY KEY AUTO_INCREMENT,
                    order_id INT NOT NULL,
                    variant_id INT NOT NULL,
                    store_id INT NOT NULL,
                    quantity INT,
                    unit_price DECIMAL(10,2),
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP C
```

```
                    FOREIGN KEY (order_id) REFERENCES orders(order_id),
                    FOREIGN KEY (variant_id) REFERENCES product_variants(v
                    FOREIGN KEY (store_id) REFERENCES stores(store_id)
);

CREATE TABLE order_fees (
                    order_fee_id INT PRIMARY KEY AUTO_INCREMENT,
                    order_id INT NOT NULL,
                    fee_type VARCHAR(50),
                    amount DECIMAL(10,2),
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON
                    FOREIGN KEY (order_id) REFERENCES orders(order_id)
);

CREATE TABLE order_discounts (
                     order_discount_id INT PRIMARY KEY AUTO_INCREMENT,
                     order_fee_id INT NOT NULL,
                     voucher_id INT NOT NULL,
                     discount_amount DECIMAL(10,2),
                     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                     updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                     FOREIGN KEY (order_fee_id) REFERENCES order_fees(ord
                     FOREIGN KEY (voucher_id) REFERENCES vouchers(vouch
);

CREATE TABLE product_images (
                     image_id SERIAL PRIMARY KEY,
                     product_id INT NOT NULL,
                     image_url TEXT NOT NULL,
                     is_main BOOLEAN DEFAULT FALSE,
                     FOREIGN KEY (product_id) REFERENCES products(product
);
```

- In designing a relational database for a ecommerce store system, I approach 3 normalization, that’s 1nf 2nf and 3nf. Above this description short, example to table is users, addresses, products, orders:
    - Normalization 1 (1NF):

        - Request: Each column contain value and table have primary key.
        - Example: Addresses table meet requirements 1NF because:
            - Each column (street, city, postal_code) contains only one value (e.g., street = "123 Test Street", not a list of addresses).
            - Have primary key addresses_id.
    - Normalization 2 (2NF):
        - Request: Meet 1NF and each column don’t dependencies to primary key.
        - Example: order_details table have single primary order_detail_id. Columns like quantity, unit_price, variant_id depend entirely on order_detail_id, not on any other part. The product name is not stored in order_details but linked through variant_id to product_variants, avoiding partial dependencies.
    - Normalization 3 (3NF):
        - Request: Meet 2NF and no non-key column depends on another non-key column.
        - Example: The addresses table is separated from users to store address information (street, city, postal_code), linked via user_id. If street is stored directly in users, when a user has multiple addresses, data will be duplicated (e.g., duplicate email for each address). Separating the addresses table helps eliminate transitive dependencies, only storing user_id in orders for linking, avoiding redundancy. Similarly, the products table uses category_id to link to categories instead of storing category_name, reducing duplication.
- Conclusion: The database achieves 3NF, with the addresses table being a typical example of data separation to avoid duplication, ensure integrity and ease of maintenance.

Question B: User "assessment", with information as shown, purchased the product "KAPPA Women's Sneakers" in yellow, size 36, quantity 1. Please write a query to insert the order that this person purchased database.

```
INSERT INTO provinces (province_id, name)
VALUES (1, 'Bắc Kạn');

INSERT INTO districts (district_id, name, province_id)
VALUES (1, 'Ba Bể', 1);

INSERT INTO communes (commune_id, name, district_id)
VALUES (1, 'Phúc Lộc', 1);

INSERT INTO housing_types (housing_type_id, name)
VALUES (1, 'Nhà riêng');

INSERT INTO users (name, email, phone)
VALUES ('assessment', 'gu@gmail.com', '328355333');

INSERT INTO addresses (user_id, commune_id, street, housing_type_id)
SELECT user_id, 1, '73 tân hòa 2', 1
FROM users
WHERE email = 'gu@gmail.com';

INSERT INTO categories (name)
VALUES ('Giày nữ');

INSERT INTO brands (name)
VALUES ('KAPPA');

INSERT INTO products (name, category_id, brand_id, base_price, gender)
VALUES ('KAPPA Women''s Sneakers',
      (SELECT category_id FROM categories WHERE name = 'Giày nữ'),
      (SELECT brand_id FROM brands WHERE name = 'KAPPA'),
      980000, 'Nữ');

INSERT INTO product_variants (product_id, color, size, stock_quantity, price_a
SELECT product_id, 'yellow', '36', 1, 0
FROM products
WHERE name = 'KAPPA Women''s Sneakers';

INSERT INTO stores (name, location, description)
```

```
VALUES ('KAPPA Store - Nguyễn Huệ', '52 Nguyễn Huệ, Quận 1, TP.HCM', 'Ng

INSERT INTO product_stocks (product_id, store_id, quantity_on_hand)
SELECT p.product_id, s.store_id, 1
FROM products p
        CROSS JOIN stores s
WHERE p.name = 'KAPPA Women''s Sneakers'
  AND s.name = 'KAPPA Store - Nguyễn Huệ';

INSERT INTO orders (user_id, address_id, order_date, payment_method, total_
SELECT u.user_id, a.address_id, NOW(), 'COD', 980000
FROM users u
        JOIN addresses a ON u.user_id = a.user_id
WHERE u.email = 'gu@gmail.com';

INSERT INTO order_details (order_id, variant_id, store_id, quantity, unit_price)
SELECT o.order_id, pv.variant_id, s.store_id, 1, 980000
FROM orders o
        JOIN users u ON o.user_id = u.user_id
        JOIN product_variants pv ON pv.color = 'yellow' AND pv.size = '36'
        JOIN products p ON pv.product_id = p.product_id
        JOIN stores s ON s.name = 'KAPPA Store - Nguyễn Huệ'
WHERE u.email = 'gu@gmail.com'
  AND p.name = 'KAPPA Women''s Sneakers'
  AND o.order_date = (SELECT MAX(order_date) FROM orders WHERE user_id
```

Question C: Write a query to calculate the average order value (total price of items in an order) for each month in the current year.

```
SELECT ROUND(AVG(object_value)), MONTH(month_in_year)
FROM (
        SELECT SUM(od.quantity * od.unit_price) AS object_value, o.order_date
        FROM orders o
                JOIN order_details od ON o.order_id = od.order_id
        WHERE YEAR(o.order_date) = YEAR(CURDATE())
        GROUP BY o.order_date
```

```
    ) AS ovmiy
GROUP BY MONTH(month_in_year)
ORDER BY MONTH(month_in_year) ASC;
```

Question D: Write a query to calculate the churn rate of customers. The churn rate is defined as the percentage of customers who did not make a purchase in the last 6 months but had made a purchase in the 6 months prior to that

```
WITH ChurnedCustomers AS (
   SELECT COUNT(DISTINCT old_customer) AS count_customer_buy_in_12mo
   FROM (
         SELECT DISTINCT u.user_id AS old_customer
         FROM users u
               LEFT JOIN orders o ON u.user_id = o.user_id
            AND o.order_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
         WHERE o.user_id IS NULL
      ) AS churn_candidate
         JOIN orders o2 ON o2.user_id = churn_candidate.old_customer
   WHERE o2.order_date BETWEEN DATE_SUB(CURDATE(), INTERVAL 12 MON
          AND DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
),
    TotalCustomers AS (
      SELECT COUNT(DISTINCT user_id) AS total_customers
      FROM orders
      WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 12 MONTH)
   )
SELECT
   ROUND((churned.count_customer_buy_in_12month / total_customers.total_c
FROM ChurnedCustomers churned
      CROSS JOIN TotalCustomers total_customers
WHERE total_customers.total_customers > 0;
```

Question E: Write API specifications, API according to RESTful standards to:
- 1) Fetches a list of all product categories available in the e-commerce
platform.
- 2) Fetches a list of products that belong to a specific category.
- 3) Allows users to search (full-text search) for products using various filters and
search terms.
- 4) Creates a new order and processes payment.
- 5) Send order confirmation email to user (processed asynchronously with order
creation flow).

1. **API: Fetch All Product Categories**

- HTTP Method: GET
- URL: /api/categories
- **Description**: List of all product categories available in the e-commerce in platform.
- **Request Parameters**: None.
- **Response:**
  - Status Code:
    - 200 OK: If list all categories success.
    - 1299 Internal Server Error: If have error on server.
  - Content-Type: application/json
  - Success Response Body:

```
{
   "status": 200,
   "message": "Success",
   "data": [
      {
         "categoryId": 1,
```

```
        "name": "Giày nữ"
      }
    ]
}
```

- Error Response Body:

```
{
    "status": 1299,
    "message": "Lỗi không xác định!",
    "data": null
}
```

2. API: **Fetch Products by Category**

- HTTP Method: GET
- URL: /api/products/category/{categoryId}
- Description: Returns a list of products belonging to a specific category, supporting pagination and sorting.
- Request Parameters:
  - Path variable:
    - categoryId (int, must number): ID of category
  - Query params:
    - search (string, optional): Search term for full-text search on product name and description.
    - categoryId (integer, optional): ID of the category to filter products.
    - brandId (integer, optional): ID of the brand to filter products.
    - minPrice (decimal, optional): Minimum base price to filter products.
    - maxPrice (decimal, optional): Maximum base price to filter products.
    - page (integer, optional, default: 0): Page number for pagination.
    - size (integer, optional, default: 10): Number of records per page.

        - sort (string, optional, default: createdAt,desc): Sorting criteria (e.g., basePrice,asc or basePrice,desc).
- Request Body: None
- Response:
    - Status Code:
        - 200 OK: If list all product by category name.
        - 602 Not Found: If category not existed .
        - 104 Bad request: If invalid format.
    - Content-Type: application/json
    - Success Response Body:


```
{
   "status": 200,
   "message": "Success",
   "data": {
      "content": [
         {
            "productId": 1,
            "name": "KAPPA Women's Sneakers",
            "basePrice": 980000.00,
            "gender": "Nữ",
            "categoryName": "Giày nữ",
            "brandName": "KAPPA",
            "images": null
         }
      ],
      "pageable": {
         "pageNumber": 0,
         "pageSize": 10,
         "sort": {
            "sorted": true,
            "empty": false,
            "unsorted": false
         },
         "offset": 0,
```

```
            "paged": true,
            "unpaged": false
        },
        "last": true,
        "totalPages": 1,
        "totalElements": 1,
        "size": 10,
        "number": 0,
        "sort": {
            "sorted": true,
            "empty": false,
            "unsorted": false
        },
        "first": true,
        "numberOfElements": 1,
        "empty": false
    }
}
```


- Error Response Body:


```
{
    "status": 104,
    "message": "Định dạng không hợp lệ!",
    "data": null
}
```


3. API: **Search Products with Filters**

- HTTP Method: GET
- URL: /api/products/search
- Description: Allows users to search for products using full-text search on product name and description, with optional filters for category, brand, and price range.
- Request Parameters:
    - Query params:

        - search (string, optional): Search term for full-text search on product name and description.
        - categoryId (integer, optional): ID of the category to filter products.
        - brandId (integer, optional): ID of the brand to filter products.
        - minPrice (decimal, optional): Minimum base price to filter products.
        - maxPrice (decimal, optional): Maximum base price to filter products.
        - page (integer, optional, default: 0): Page number for pagination.
        - size (integer, optional, default: 10): Number of records per page.
        - sort (string, optional, default: createdAt,desc): Sorting criteria (e.g., basePrice,asc or basePrice,desc).
- Request Body: None
- Response:
    - Status Code:
        - 200 OK: If list all product by category name.
        - 500
    - Content-Type: application/json
    - Success Response Body:


```
{
  "status": 200,
  "message": "Success",
  "data": {
    "content": [
      {
        "productId": 1,
        "name": "KAPPA Women's Sneakers",
        "basePrice": 980000.00,
        "gender": "Nữ",
        "categoryName": "Giày nữ",
        "brandName": "KAPPA",
        "images": null
      }
    ],
```

```
    "pageable": {
      "pageNumber": 0,
      "pageSize": 10,
      "sort": {
        "sorted": true,
        "empty": false,
        "unsorted": false
      },
      "offset": 0,
      "paged": true,
      "unpaged": false
    },
    "last": true,
    "totalPages": 1,
    "totalElements": 1,
    "size": 10,
    "number": 0,
    "sort": {
      "sorted": true,
      "empty": false,
      "unsorted": false
    },
    "first": true,
    "numberOfElements": 1,
    "empty": false
  }
}
```


- Error Response Body:


```
{
  "status": 500,
  "message": "Internal server error",
  "data": null
}
```


4. API: Creates a new order and processes payment

- HTTP Method: POST
- URL: /api/orders
- Description: Creates a new order for a user, processes payment (currently supports Cash on Delivery), and updates stock quantities of product variants. The request specifies the user, delivery address, payment method, and order details, including product variants, quantities, and prices.
- Request Parameters:
  - Query params: None.
- Request Body:

```
{
  "userId": 1,
  "addressId": 1,
  "paymentMethod": "COD",
  "voucherCode": "DISCOUNT10",
  "items": [
    {
      "variantId": 1,
      "storeId": 1,
      "quantity": 2,
      "unitPrice": 980000.00
    }
  ]
}
```

- Response:
  - Status Code:
    - 201 Created: Order created, payment processed, and stock updated successfully.
    - 400 Bad Request: Invalid request data (e.g., insufficient stock, invalid payment method, negative quantity).
    - 404 Not Found: User, address, product variant, or store not found.
    - 500 Internal Server Error: Unexpected server error.
  - Content-Type: application/json

- Success Response Body:

```
{
   "status": 200,
   "message": "Success",
   "data": {
      "orderId": 14,
      "userId": 1,
      "addressId": 1,
      "orderDate": "2025-05-14T04:10:32.541800357",
      "paymentMethod": "COD",
      "totalAmount": 1960000.00,
      "status": "PENDING",
      "items": [
         {
            "variantId": null,
            "storeId": null,
            "quantity": 2,
            "unitPrice": 980000.00
         }
      ]
   }
}
```

- Error Response Body:

```
{
   "status": 400,
   "message": "Dữ liệu không hợp lệ",
   "data": {
      "userId": "ID người dùng phải là số dương"
   }
}
```

5. **API: Send order confirmation email**

- **HTTP Method**: POST

- **URL**: /api/orders
- **Description**: Creates a new order for a user, processes payment (currently supports Cash on Delivery), updates stock quantities of product variants, and sends an order confirmation email to the user asynchronously. The request specifies the user, delivery address, payment method, and order details.
- **Request Parameters**:
  - **Query params**: None.
- **Request Body**:

```
{
  "userId": 1,
  "addressId": 1,
  "paymentMethod": "COD",
  "voucherCode": "DISCOUNT10",
  "items": [
    {
      "variantId": 1,
      "storeId": 1,
      "quantity": 2,
      "unitPrice": 980000.00
    }
  ]
}
```

- Response:
  - Status Code:
    - 201 Created: Order created, payment processed, and stock updated successfully.
    - 400 Bad Request: Invalid request data (e.g., insufficient stock, invalid payment method, negative quantity).
    - 404 Not Found: User, address, product variant, or store not found.
    - 500 Internal Server Error: Unexpected server error.
  - Content-Type: application/json

- Success Response Body:

```
Kính gửi Quý Khách,

Cảm ơn bạn đã đặt hàng tại cửa hàng của chúng tôi! Dưới đây là thông

Mã Đơn Hàng: #**
Ngày Đặt Hàng: 14/05/2025 04:22:03
Phương Thức Thanh Toán: COD
Trạng Thái: PENDING

Chi Tiết Đơn Hàng:
----------------------------------------
Sản Phẩm: KAPPA Women's Sneakers (Màu: yellow, Kích Cỡ: 36)
Số Lượng: **
Đơn Giá: *** ₫
----------------------------------------
Tổng Tiền: *** ₫

Chúng tôi sẽ xử lý đơn hàng của bạn sớm nhất có thể. Nếu bạn có bất

Trân trọng,
Đội Ngũ Bán Hàng
```

Question F: Implement the above APIs with the language/framework you choose.