A detailed document for the information needed to store in memory and in the database for each business entity (like a customer or a payment). - **4 points**

Entities Stored in the Database (the data types shown below are what they represented as in memory in the model classes; everything is represented as a string in the Redis database itself):

1. **Users (for storing information about users who can buy or sell)**
    a. userID: int
    b. email: String
    c. username: String
    d. Password: String
2. **Products (for storing information about products that users can list for sale or purchase; keeps track of which user listed the product)**
    a. productID: int
    b. productName: String
    c. stock: int
    d. price: double
    e. sellerID: int
3. **Orders (for storing information about when a user purchases a product; keeps track of the product sold, the buyer, the seller, and payment information)**
    a. orderID: int
    b. productID: int
    c. buyerID: int
    d. sellerID: int
    e. orderPrice: double
    f. creditCard: String
    g. CVV: String

Also stored in the database:

- order_counter (stores the number of orders in the database so the next key for the orders is easy to calculate)

Notes about the database:

The userID is used to keep track of who is putting a product for sale as the sellerID field in the Products entity. The userID is also used to keep track of who is the seller and who is the buyer as the sellerID and buyerID fields in the Orders entity.

Unlike the relational database in Project 1, there is no Payments entity because information about the payments can just be stored in the Orders entity itself. Professor Nguyen said for this project, there can be just a single product in an order so I only store information about one product in an order. There is also no suppliers entity here since the “users” are the “suppliers” themselves as they are the ones who list the products for sale.

I stored a value with the key “order_counter” in the database which is used to track how many orders have been placed and is incremented every time an order is placed. This way I can just make the key for the next order placed to be “order:<order_counter>” and continue incrementing the order_counter.

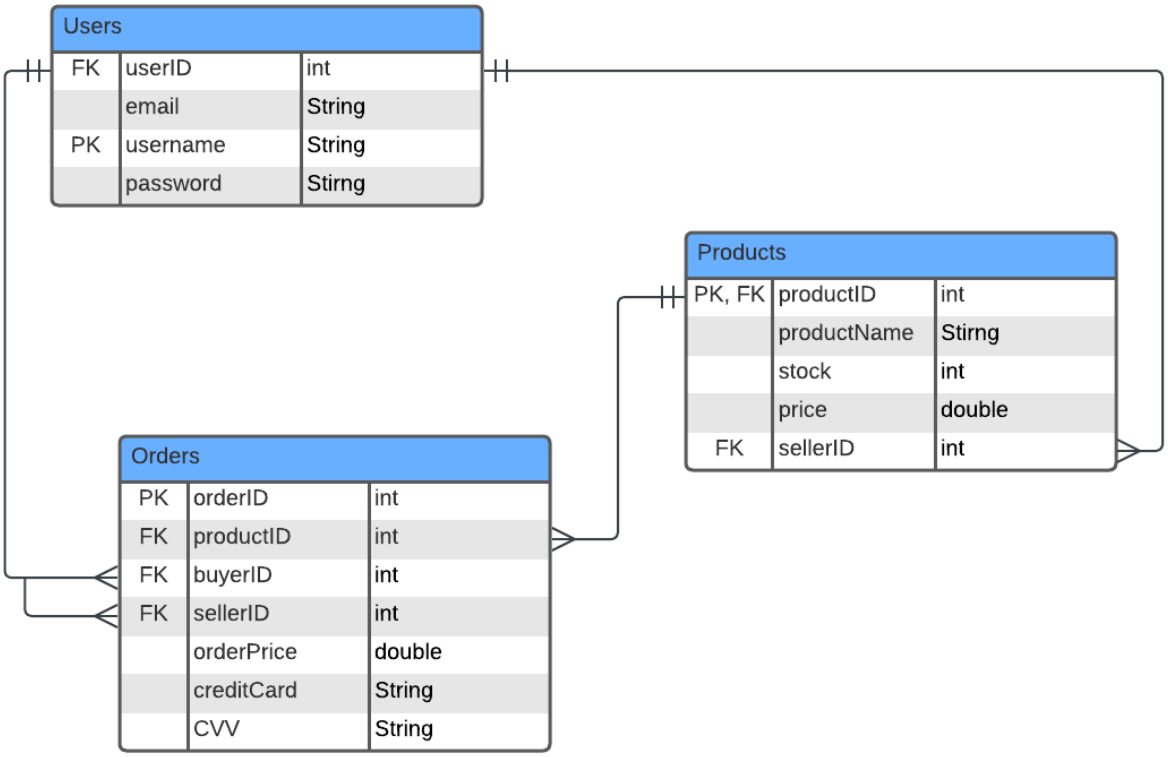
How the Information About Those Entities is Stored in Memory (i.e. Model Classes): Each of the following contains constructors, getters, and setters (but not shown here):

```
public class User {
    private int userID;
    private String email;
    private String username;
    private String password;
}

public class Product {
    private int productID;
    private String productName;
    private int stock;
    private double price;
    private int sellerID;
}

public class Order {
    private int productID;
    private int buyerID;
    private int sellerID;
    private double orderPrice;
    private String creditCard;
    private String cvv;
}
```

A database scheme. **- 4 points**
[Include an ERD diagram, which clearly defines the relationships between entities. Add description describing the ERD and relationships]



ERD Description and Relationships:

Users Relationships:

- One-to-Many with Orders - one user can make many orders (make many purchase)
- One-to-Many with Orders - one user can sell many orders (have many things purchased from them)

Products Relationships:

- Many-to-One with Users - many products can be listed for sale by one user
- One-to-Many with Orders - one product could be a part of many orders (*Dr. Nguyen said in class we did not have to include many products in a single order*)

Orders Relationships:

- Many-to-One with Products - many orders can deal with the same one product (*Dr. Nguyen said in class we did not have to include many products in a single order*)
- Many-to-One with Users - many orders can be made by one user
- Many-to-One with Users - many orders can be sold by one user