M5.7 Assignment: Create an ERD

By Sheikh Uddin

Overview

In this assignment, you will design an Entity-Relationship Diagram (ERD) to model the structure of a system of your choice. The ERD should reflect the logical design stage and include entities, relationships, attributes, primary and foreign keys, and data types. You are free to choose the type of system you want to model (e.g., an online store, a healthcare management system, a student course enrollment system), but the ERD must meet the minimum requirements listed below.

Minimum Requirements

- Entities
  - Include at least 3 entities.
  - Each entity must have a minimum of 4 attributes, including the primary key.
  - Each entity must have a Primary Key (PK).
- Attributes
  - Use at least 3 different data types across all entities (e.g., INT, DECIMAL(p, s), FLOAT, CHAR(n), VARCHAR(n), TEXT, DATE, TIME, DATETIME, TIMESTAMP, BOOLEAN).
- Relationships
  - Establish relationships between entities with appropriate cardinality.
  - Include at least 2 foreign keys (FK) to represent these relationships.
  - Use Crow's Foot notation to represent relationships in your ERD.

Steps

Choose a System to Model

Select a system that you are familiar with or interested in.

Examples include but are not limited to:

- Online Retail Store
- Hospital Management System
- Social Media Platform
- University Enrollment System
- Hotel Reservation System
- Library Management System

- Inventory Management System
- Any other system of your choice

Define Entities and Attributes

- List all entities and their attributes.
- Assign data types to each attribute.
- Identify primary keys for each entity.
- Example:
    - Customers
        - CustomerID (INT) (PK)
        - FirstName (VARCHAR(50))
        - LastName (VARCHAR(50))
        - Email (VARCHAR(100))
    - Orders
        - OrderID (INT) (PK)
        - Customer ID (INT)
        - Ship Date (DATE)

Establish Relationships and Foreign Keys

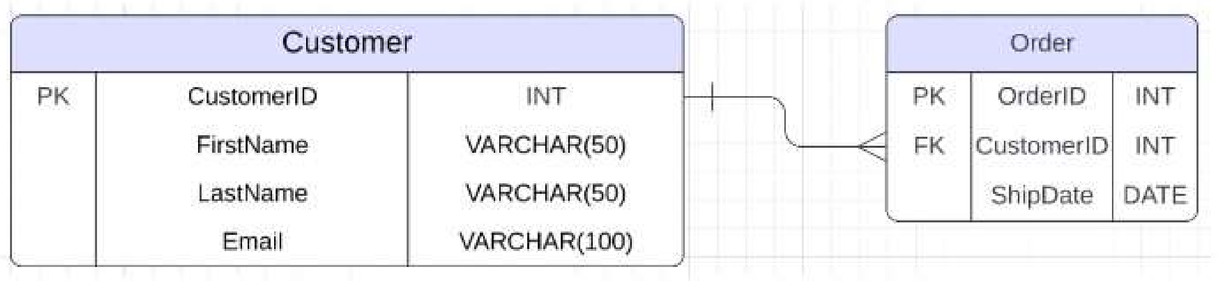
- Determine how entities are related.
- Define foreign keys to establish these relationships.
- Specify cardinality for each relationship.
- Example:
    - Relationship:
        - Customer places Order
    - Cardinality:
        - One-to-Many (One Customer can place many Orders)
    - Foreign Key connecting Order Entity with Customer Entity
        - Orders
            - CustomerID (INT) [FK]

Create the ERD

Use Lucidchart to draw the ERD of your system. Use the included ERD template to draw:

- Entities Attributes listed inside entities
- Data types
- Primary Keys (marked as PK)
- Foreign Keys (marked as FK)
- Relationships drawn as lines, using Crow's Foot notation to indicate cardinality

Example:



Deliverables

A PDF document containing:

- An image of your ERD
- A link to the ERD on Lucidchart
- A brief description (1-2paragraphs) explaining:
  - Each entity and its role within the system.
  - The relationships between entities and the reasoning behind the cardinality chosen.
  - Any assumptions made during the design process.

Resources

- Lucidchart Tutorial Video showing general construction process, template usage, and getting sharable linkLinks to an external site.
- Lucidchart tutorial video showing template use and cardinality settingLinks to an external site.

Steps 1:

Choose a System to Model

Online Retail Store

Introduction:

The Online Retail Store System ERD represents a database design for an e-commerce platform where customers can browse and purchase products.

Steps 2:

Define Entities and Attributes

List all entities and their attributes.

---

1. **CUSTOMERS**

- **CustomerID** (INT, PK): Unique identifier for each customer.
- **FirstName** (VARCHAR(50)): Customer's first name.
- **LastName** (VARCHAR(50)): Customer's last name.
- **Address** (VARCHAR(255)): Customer's full address.
- **Email** (VARCHAR(100)): Customer's email address.
- **City** (VARCHAR(100)): City of the customer.
- **State** (VARCHAR(100)): State of the customer.
- **ZipCode** (VARCHAR(20)): Postal code for the customer's address.
- **Country** (VARCHAR(100)): Customer's country of residence.
- **PhoneNumber** (VARCHAR(15)): Customer's contact phone number.

---

2. **CATEGORIES**

- **CategoryID** (INT, PK): Unique identifier for each product category.
- **CategoryName** (VARCHAR(100)): Name of the product category.
- **Description** (TEXT): Description of the product category.
- **CreatedDate** (DATETIME): Date and time the category was created (default: current timestamp).
- **IsActive** (BOOLEAN): Indicates if the category is currently active (default: TRUE).

---

3. **PRODUCTS**

- **ProductID** (INT, PK): Unique identifier for each product.
- **ProductName** (VARCHAR(100)): Name of the product.
- **Description** (TEXT): Detailed description of the product.
- **Price** (DECIMAL): Price of the product.
- **QuantityInStock** (INT): Current stock level of the product.
- **CategoryID** (INT, FK): Foreign key that references the `CATEGORIES(CategoryID)`.

- **ImageURL** (VARCHAR(255)): URL to an image of the product.

---

4. **ORDERS**

- **OrderID** (INT, PK): Unique identifier for each order.
- **OrderDate** (DATETIME): Date and time when the order was placed.
- **OrderStatus** (VARCHAR(20)): Status of the order (e.g., Pending, Shipped, Completed).
- **TotalAmount** (DECIMAL): Total amount for the order.
- **CustomerID** (INT, FK): Foreign key that references the `CUSTOMERS(CustomerID)`.
- **ShippingAddressID** (INT): Placeholder for shipping address (not defined as a foreign key in this design).
- **ReturnAddressID** (INT): Placeholder for return address (not defined as a foreign key in this design).

---

5. **ORDERITEMS**

- **OrderItemID** (INT, PK): Unique identifier for each item in an order.
- **OrderID** (INT, FK): Foreign key that references the `ORDERS(OrderID)`.
- **ProductID** (INT, FK): Foreign key that references the `PRODUCTS(ProductID)`.
- **Quantity** (INT): Number of units of the product ordered.
- **Price** (DECIMAL): Price of the product at the time of the order (snapshot of `PRODUCTS.Price`).

---

6. **PAYMENTS**

- **PaymentID** (INT, PK): Unique identifier for each payment transaction.
- **OrderID** (INT, FK): Foreign key that references the `ORDERS(OrderID)`.
- **PaymentMethod** (VARCHAR(50)): Payment method used (e.g., Credit Card, PayPal).
- **PaymentDate** (DATETIME): Date and time the payment was made.
- **Amount** (DECIMAL): Amount paid.

---

7. **SHIPPING**

- **ShippingID** (INT, PK): Unique identifier for each shipping record.
- **OrderID** (INT, FK): Foreign key that references the `ORDERS(OrderID)`.
- **ShippingCarrier** (VARCHAR(100)): Shipping carrier (e.g., FedEx, UPS).

- **TrackingNumber** (VARCHAR(100)): Tracking number provided by the carrier.
- **EstimatedDeliveryDate** (DATE): Estimated date of delivery.

---

8. **RETURNS**

- **ReturnID** (INT, PK): Unique identifier for each return transaction.
- **OrderID** (INT, FK): Foreign key that references the `ORDERS(OrderID)`.
- **ReturnReason** (VARCHAR(255)): Reason for the return.
- **ReturnDate** (DATE): Date the return was initiated.
- **ReturnStatus** (VARCHAR(50)): Status of the return (e.g., Approved, Rejected, Processing).

---

9. **REVIEWS**

- **ReviewID** (INT, PK): Unique identifier for each review.
- **ProductID** (INT, FK): Foreign key that references the `PRODUCTS(ProductID)`.
- **CustomerID** (INT, FK): Foreign key that references the `CUSTOMERS(CustomerID)`.
- **Rating** (INT): Rating provided by the customer (e.g., on a scale of 1 to 5).
- **Comment** (TEXT): Customer's review comment or feedback.

Steps 3:

Establish Relationships and Foreign Keys

Determine how entities are related.

---

1. **Relationship**:

Customer places Order

- **Cardinality**:
  One-to-Many (One Customer can place many Orders)
- **Foreign Key connecting Order Entity with Customer Entity**:
  Orders
  `CustomerID` (INT) [FK]

---

2. **Relationship**:

Category contains Product

- **Cardinality**:
  One-to-Many (One Category can contain many Products)
- **Foreign Key connecting Product Entity with Category Entity**:
  Products
  `CategoryID` (INT) [FK]

---

3. **Relationship**:

Order contains OrderItem

- **Cardinality**:
  One-to-Many (One Order can have many OrderItems)
- **Foreign Key connecting OrderItem Entity with Order Entity**:
  OrderItems
  `OrderID` (INT) [FK]

---

4. **Relationship**:

Product appears in OrderItem

- **Cardinality**:
  One-to-Many (One Product can appear in many OrderItems)
- **Foreign Key connecting OrderItem Entity with Product Entity**:
  OrderItems
  `ProductID` (INT) [FK]

---

5. **Relationship**:

Order has Payment

- **Cardinality**:
  One-to-Many (One Order can have many Payments)
- **Foreign Key connecting Payment Entity with Order Entity**:
  Payments
  `OrderID` (INT) [FK]

---

6. **Relationship**:

Order has Shipping

- **Cardinality**:
  One-to-One (One Order can have only one Shipping record)
- **Foreign Key connecting Shipping Entity with Order Entity**:
  Shipping
  `OrderID` (INT) [FK]

---

7. **Relationship**:

Order has Return

- **Cardinality**:
  One-to-Many (One Order can have many Returns)
- **Foreign Key connecting Return Entity with Order Entity**:
  Returns
  `OrderID` (INT) [FK]

---

8. **Relationship**:

Product receives Review

- **Cardinality**:
  One-to-Many (One Product can receive many Reviews)
- **Foreign Key connecting Review Entity with Product Entity**:
  Reviews
  `ProductID` (INT) [FK]

---

9. **Relationship**:

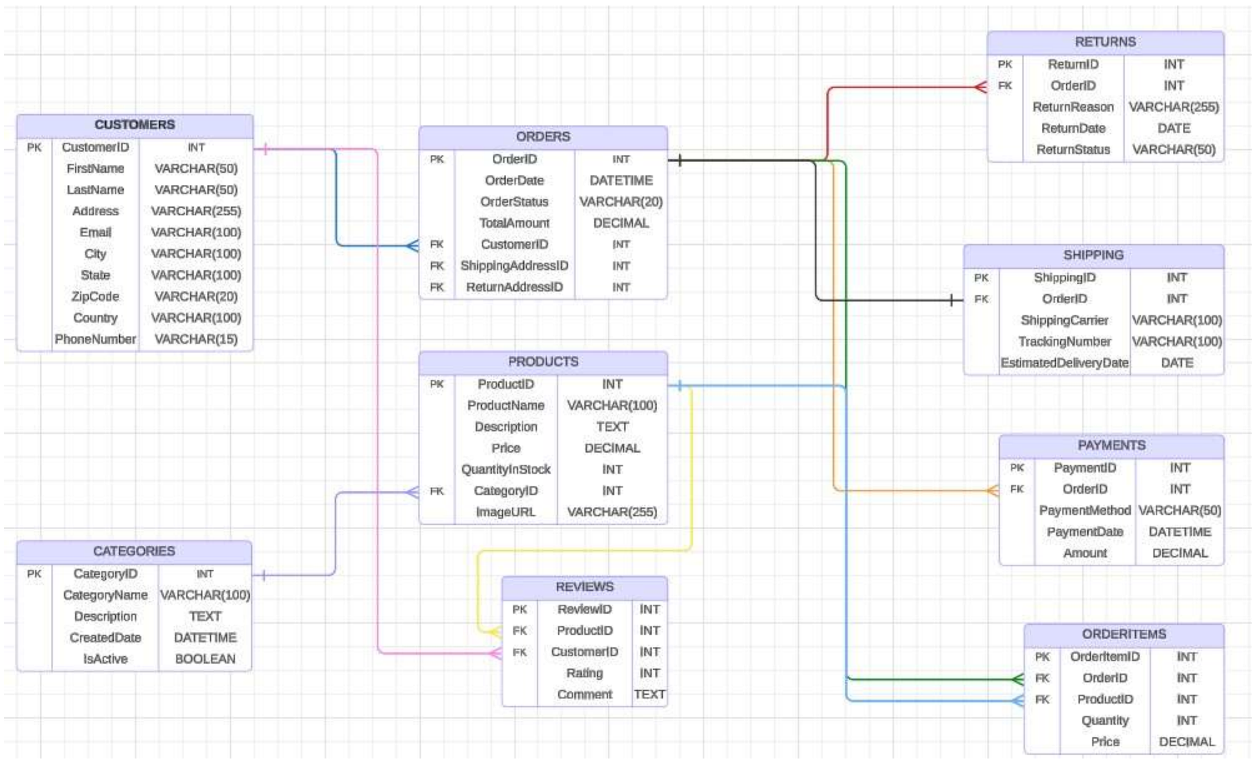
Customer writes Review

- **Cardinality**:
  One-to-Many (One Customer can write many Reviews)
- **Foreign Key connecting Review Entity with Customer Entity**:
  Reviews
  `CustomerID` (INT) [FK]

Steps 4:

Create the ERD

Use Lucidchart to draw the ERD of your system. Use the included ERD template to draw:



A link to the ERD on Lucidchart

1:Lucidchart Link:

https://lucid.app/lucidchart/6183e454-b841-4ead-b332-a5388b958a1d/edit?viewport_loc=-2198%2C-1266%2C3508%2C1520%2Cephget3akdC8&invitationId=inv_6ab7bd62-3fb1-4b94-9171-9cf930dea79a

2:Test ERD BY sqlfiddle:

https://sqlfiddle.com/mysql/online-compiler?id=a20c9bbb-2b9c-4e8f-a271-2ac2ce7e04a9

A brief description (1-2paragraphs) explaining

The relationships between entities and the reasoning behind the cardinality chosen.

1. **CUSTOMERS and ORDERS**

- **Relationship**: A customer can place multiple orders, but each order is placed by only one customer.
- **Cardinality**: **One-to-Many (1:N)**
  - **Reasoning**: A single customer (1) can make many orders (N), but each order is linked to one and only one customer.
  - **Foreign Key**: `CustomerID` in the `ORDERS` table references `CustomerID` in the `CUSTOMERS` table.

2. **ORDERS and ORDERITEMS**

- **Relationship**: An order can contain multiple products (items), and each product is part of one specific order.
- **Cardinality**: **One-to-Many (1:N)**
  - **Reasoning**: An order (1) typically contains multiple products (N), but each order item belongs to one specific order.
  - **Foreign Key**: `OrderID` in the `ORDERITEMS` table references `OrderID` in the `ORDERS` table.

3. **ORDERITEMS and PRODUCTS**

- **Relationship**: A product can appear in multiple order items (since different customers can buy the same product), and each order item is associated with one product.
- **Cardinality**: **Many-to-One (N:1)**
  - **Reasoning**: Multiple order items (N) can refer to the same product (1) because the same product can be ordered by different customers in different orders.
  - **Foreign Key**: `ProductID` in the `ORDERITEMS` table references `ProductID` in the `PRODUCTS` table.

4. **CATEGORIES and PRODUCTS**

- **Relationship**: A category can have multiple products, but each product belongs to one specific category.
- **Cardinality**: **One-to-Many (1:N)**
  - **Reasoning**: A single category (1) can contain many products (N), but each product belongs to only one category.
  - **Foreign Key**: `CategoryID` in the `PRODUCTS` table references `CategoryID` in the `CATEGORIES` table.

5. **ORDERS and PAYMENTS**

- **Relationship**: An order can have multiple payments (if partial payments are allowed), but each payment is associated with one order.
- **Cardinality**: **One-to-Many (1:N)**
  - **Reasoning**: One order (1) might have multiple payments (N), such as in cases of partial payments or installment plans, but each payment relates to only one order.
  - **Foreign Key**: `OrderID` in the `PAYMENTS` table references `OrderID` in the `ORDERS` table.

6. ORDERS and SHIPPING

- **Relationship**: An order can have one shipping record, and a shipping record is associated with one order.
- **Cardinality**: **One-to-One (1:1)**
  - **Reasoning**: Each order (1) has exactly one shipping record (1) as it is shipped to the customer. Shipping details belong to the specific order.
  - **Foreign Key**: `OrderID` in the `SHIPPING` table references `OrderID` in the `ORDERS` table.

7. ORDERS and RETURNS

- **Relationship**: An order can have multiple return records (if multiple returns are allowed), but each return is associated with one order.
- **Cardinality**: **One-to-Many (1:N)**
  - **Reasoning**: One order (1) can be returned partially or fully, so there can be multiple return events (N) for an order. However, each return corresponds to one specific order.
  - **Foreign Key**: `OrderID` in the `RETURNS` table references `OrderID` in the `ORDERS` table.

8. PRODUCTS and REVIEWS

- **Relationship**: A product can have multiple customer reviews, but each review is for one specific product.
- **Cardinality**: **One-to-Many (1:N)**
  - **Reasoning**: A single product (1) can receive many reviews (N), but each review is for only one product.
  - **Foreign Key**: `ProductID` in the `REVIEWS` table references `ProductID` in the `PRODUCTS` table.

9. CUSTOMERS and REVIEWS

- **Relationship**: A customer can write multiple reviews, but each review is written by only one customer.

- **Cardinality**: **One-to-Many (1:N)**
    - **Reasoning**: A customer (1) can write many reviews (N), but each review belongs to only one customer.
    - **Foreign Key**: `CustomerID` in the `REVIEWS` table references `CustomerID` in the `CUSTOMERS` table.

Cardinality Summary:

- **One-to-Many (1:N)**:
    - **CUSTOMERS to ORDERS**: One customer can place multiple orders.
    - **ORDERS to ORDERITEMS**: One order can contain multiple items.
    - **CATEGORIES to PRODUCTS**: One category can contain multiple products.
    - **ORDERS to PAYMENTS**: One order can have multiple payments.
    - **ORDERS to RETURNS**: One order can have multiple returns.
    - **PRODUCTS to REVIEWS**: One product can have multiple reviews.
    - **CUSTOMERS to REVIEWS**: One customer can write multiple reviews.
- **Many-to-One (N:1)**:
    - **ORDERITEMS to PRODUCTS**: Many order items can refer to the same product.
- **One-to-One (1:1)**:
    - **ORDERS to SHIPPING**: Each order has one shipping record.

Reasoning Behind Cardinality Choices:

- **Customer behavior**: Customers typically place multiple orders over time, leading to a 1:N relationship with orders. Similarly, customers write multiple reviews for different products, establishing a 1:N relationship.
- **Order structure**: An order can contain multiple items (products), leading to 1:N relationships between orders and order items. Payments are typically linked to orders in a 1:N manner if partial or multiple payments are allowed.
- **Product reviews**: Products often receive multiple reviews, establishing a 1:N relationship between products and reviews. Customers can review different products, leading to another 1:N relationship.
- **Inventory management**: Products are categorized to make browsing easier, and each category can contain many products, but each product belongs to a specific category, leading to a 1:N relationship.

Any assumptions made during the design process.

1. **Customers place multiple orders**

   - **Assumption**: A single customer can place many orders over time, but each order belongs to one and only one customer. This assumption is typical for e-commerce platforms where customers may return to make multiple purchases.
   - **Impact**: This led to the **1:N relationship** between `CUSTOMERS` and `ORDERS`.

2. **Each order contains multiple items**

   - **Assumption**: Orders will often contain multiple products, with each product having a specific quantity. This reflects typical scenarios where customers purchase multiple items in a single transaction.
   - **Impact**: This drove the **1:N relationship** between `ORDERS` and `ORDERITEMS`.

3. **Products belong to only one category**

   - **Assumption**: Each product is assigned to only one category. It assumes that products won't belong to multiple categories at once, simplifying category management.
   - **Impact**: This led to a **1:N relationship** between `CATEGORIES` and `PRODUCTS`.

4. **Each order has one shipping record**

   - **Assumption**: Each order is shipped only once, and shipping information is captured in a single record. This simplifies the shipping tracking process and ensures that every order has one shipping entry.
   - **Impact**: This led to a **1:1 relationship** between `ORDERS` and `SHIPPING`.

5. **One payment method per payment**

   - **Assumption**: Each payment is made using a single payment method. If an order allows for multiple payments (e.g., in installments or partial payments), each payment is recorded separately, with its own payment method.
   - **Impact**: This resulted in a **1:N relationship** between `ORDERS` and `PAYMENTS`, allowing flexibility in handling multiple payments for one order.

6. **Each order can have multiple returns**

- **Assumption**: Customers may return one or more items from an order, leading to multiple return records for a single order (e.g., a customer returns items in different batches).
- **Impact**: This assumption led to the **1:N relationship** between `ORDERS` and `RETURNS`.

7. **Each product can receive multiple reviews**

- **Assumption**: Products typically receive reviews from multiple customers. However, each review is linked to a specific customer and product.
- **Impact**: This formed the **1:N relationship** between `PRODUCTS` and `REVIEWS`.

8. **Customers can review multiple products**

- **Assumption**: A customer may review different products. Each review is uniquely tied to one customer and one product, and a customer can submit multiple reviews for different products.
- **Impact**: This created the **1:N relationship** between `CUSTOMERS` and `REVIEWS`.

9. **Product stock is managed per product**

- **Assumption**: Each product's stock is managed individually. There's no need for a separate inventory system outside of the `PRODUCTS` table for tracking stock quantities.
- **Impact**: The `PRODUCTS` table includes a `QuantityInStock` field, assuming that stock levels are managed at the product level.

10. **Orders are distinct from customers' shipping and return addresses**

- **Assumption**: The `ShippingAddressID` and `ReturnAddressID` fields in the `ORDERS` table point to separate potential address records (not defined in the current design), assuming that customers can choose different addresses for shipping and returns.
- **Impact**: This allows flexibility for customers to use different addresses for different purposes.

11. **No need for soft deletes or historical records**

- **Assumption**: The design does not account for historical data retention (e.g., soft deletes or archiving old records). It assumes that once a record is deleted or updated, the previous state is not retained.

- **Impact**: The design focuses on a straightforward relational model without audit trails or historical tracking.

12. **All data inputs are valid**

- **Assumption**: The design assumes data validation and integrity will be handled at the application level (outside the database). For example, it assumes that product prices will not be negative and that the number of items ordered will always be positive.
- **Impact**: This assumption simplifies the schema by excluding constraints like CHECK constraints for non-negative prices.

Potential Modifications for Further Assumptions:

1. **Address management**: If the system needs to support multiple addresses per customer (billing, shipping, etc.), a separate `ADDRESSES` table might be introduced.
2. **Multi-category products**: If products can belong to multiple categories, a junction table (e.g., `PRODUCT_CATEGORY`) would need to be introduced to support a many-to-many relationship.
3. **Soft Deletes**: For audit and traceability purposes, soft delete mechanisms (e.g., `IsDeleted` flag) could be introduced to prevent physical deletion of important records.
4. **Product variants**: If the system needs to manage product variants (e.g., size, color), an additional `PRODUCT_VARIANTS` table might be required.
5. **Discount and Promotions Management**:
   Add `DISCOUNTS` and `OrderDiscounts` tables to manage product or order-based discounts, including time-bound promotions.
6. **Product Inventory Tracking**: Introduce a `STOCK_HISTORY` table to track stock changes over time and a `WAREHOUSES` table with `WAREHOUSE_PRODUCTS` for managing multiple storage locations.
7. **Order Fulfillment Status**: Create a `FULFILLMENTS` table to handle partial order shipments and track the fulfillment status for each item.
8. **Loyalty Programs**: Add a `LOYALTY_PROGRAM` table to manage customer points, tiers, and rewards based on purchasing behaviors.
9. **Customer Preferences and History**: Introduce a `CUSTOMER_PREFERENCES` table for storing customer preferences and a `PURCHASE_HISTORY` table to track past orders and behaviors.

10. **Order Lifecycle and Status Tracking**:
    Add `ORDER_HISTORY` and `SHIPPING_HISTORY` tables to maintain an audit trail of order status changes and shipping updates.
11. **Payments and Refunds**: Introduce a `REFUNDS` table for tracking refunds and a `PARTIAL_PAYMENTS` table to manage installment-based payments.
12. **Multi-Currency Support**: Add a `CURRENCY` table to manage multi-currency transactions with exchange rates for `ORDERS` and `PAYMENTS`.
13. **Customer Segmentation**: Create a `CUSTOMER_SEGMENTS` table to group customers based on various criteria like spending habits for targeted marketing.
14. **Vendor and Supplier Management**: Add a `VENDORS` table to manage supplier information and a `PURCHASE_ORDERS` table for tracking orders from suppliers, alongside a `VENDOR_PRODUCTS` table to track the products supplied.

These modifications ensure the system is scalable and adaptable to complex business needs like discounting, inventory tracking, multi-currency, advanced customer management and etc.