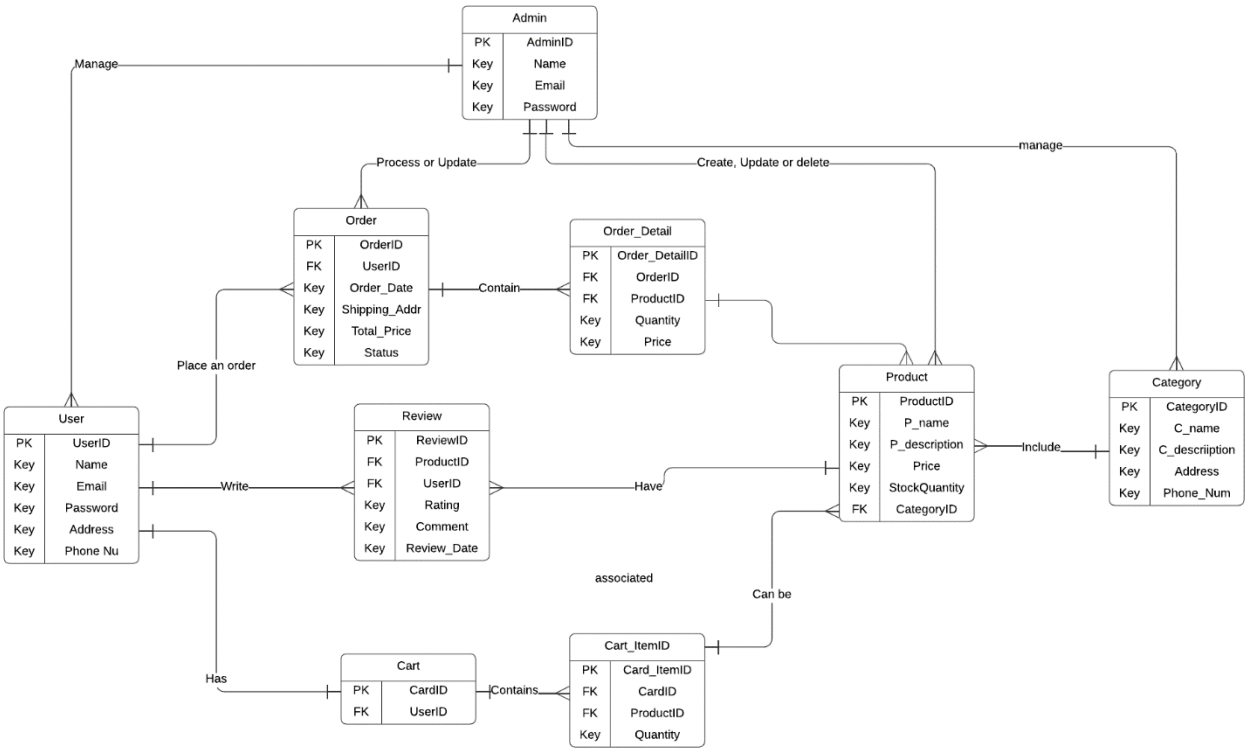
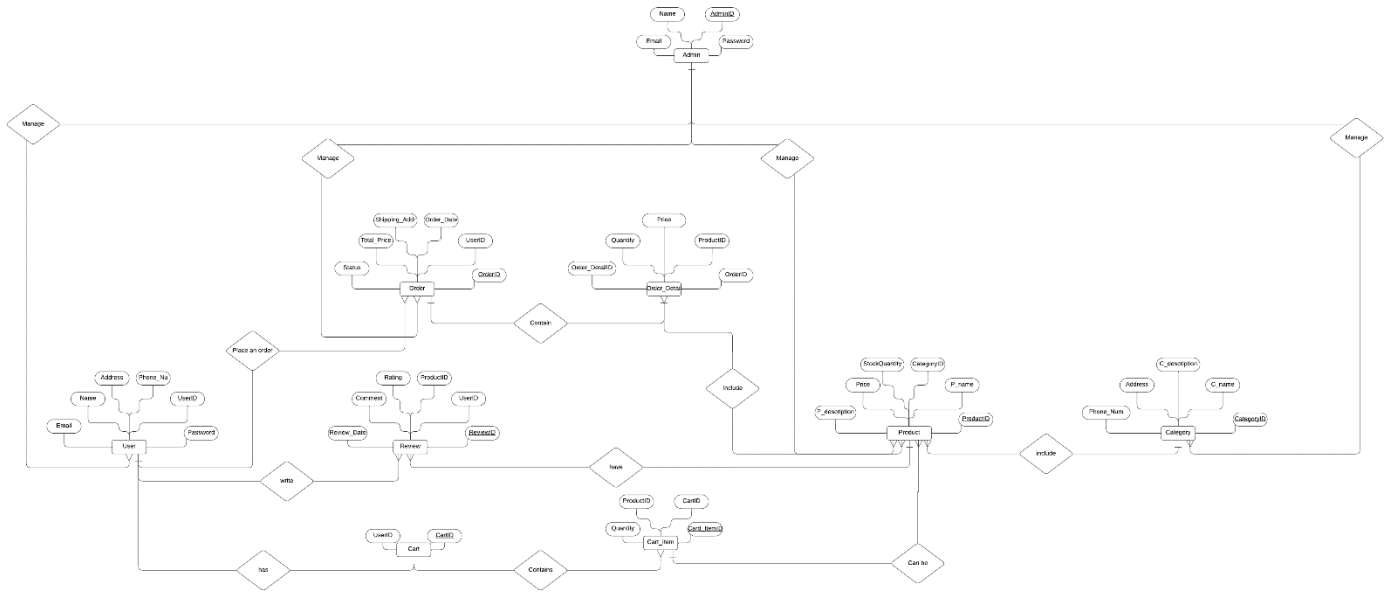
Database Design for Final Year Project

1.Entity Relation Diagram:



2.Relational Schema



3.Normalization:

In order to reduce duplication and preserve data integrity, we want to make sure that my e-commerce database is both efficient and standardized. We began by using the First Normal Form (1NF) principles, which included making sure that there were no repeated groups or arrays that would add needless complexity, and that each table had a primary key that could be used to uniquely identify each record.

As we moved to the Second Normal Form (2NF), we ensured that no attribute in any table having a composite primary key depended on more than half of the key. To ensure that all data in the database depends on the whole primary key, for instance, in my Order_Detail table, Quantity and Price depend on the full composite key of OrderID and ProductID.

We next carefully examined the tables for transitive dependencies—a situation in which non-key characteristics depend on one another—and removed them in order to achieve the Third Normal Form (3NF). This normalization phase is essential since it improves data integrity and further minimizes redundancy. One way WEdid this was by keeping user contact information in the User table instead of the Order table, which uses the UserID as a point of reference but does not retain the information directly.

We also gave the Boyce-Codd Normal Form (BCNF), a more stringent variation of the 3NF, some thought. In order to deal with any anomalies and further guarantee the removal of redundancy, we made sure that each determinant is a candidate key.

My objective was to preserve pragmatism while streamlining efficiency in the database design. WEavoided having duplicate data for each item by separating orders from their information. By creating a specific Review table, it was possible to have product feedback without duplicating product data. Together, the Cart and Cart_Item tables enable several things to be held in a single cart without duplication of data. In order to facilitate straightforward adjustments to category descriptions without affecting specific products, WEalso made a distinction between the Product and Category tables. The use of foreign keys, such as ProductID in the Order_Detail database, is essential because it prevents data duplication and maintains referential integrity.

4.Data Dictionary:

User Table

Stores information about the users. Each user has a unique UserID.

```
CREATE TABLE User (
  UserID INT PRIMARY KEY AUTO_INCREMENT,
  Name VARCHAR(255),
  Email VARCHAR(255) UNIQUE,
  Password VARCHAR(255),
```

Address VARCHAR(255),

Phone_Num VARCHAR(255)

);

Admin Table

Stores information about the admins.Each admin has a unique AdminID.

CREATE TABLE Admin (

AdminID INT PRIMARY KEY AUTO_INCREMENT,

Name VARCHAR(255),

Email VARCHAR(255) UNIQUE,

Password VARCHAR(255)

);

Order Table

Contains details about orders placed by users. Each order has a unique OrderID. Linked to User via UserID.

CREATE TABLE Order (

OrderID INT PRIMARY KEY AUTO_INCREMENT,

UserID INT,

Order_Date DATETIME,

Shipping_Addr VARCHAR(255),

Total_Price DECIMAL(10, 2),

Status VARCHAR(255),

FOREIGN KEY (UserID) REFERENCES User(UserID)

);

Order_Detail Table

Contains the details of the products within each order. Each order detail has a unique Order_DetailID.Linked to Order via OrderID and Product via ProductID.

CREATE TABLE Order_Detail (

Order_DetailID INT PRIMARY KEY AUTO_INCREMENT,

OrderID INT,

```
    ProductID INT,
    Quantity INT,
    Price DECIMAL(10, 2),
    FOREIGN KEY (OrderID) REFERENCES Order(OrderID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

Product Table.

Information about products.Each product has a unique ProductID.Linked to Category via CategoryID.

```
CREATE TABLE Product (
    ProductID INT PRIMARY KEY AUTO_INCREMENT,
    P_name VARCHAR(255),
    P_description TEXT,
    Price DECIMAL(10, 2),
    StockQuantity INT,
    CategoryID INT,
    FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID)
);
```

Category Table

Information about the categories of products. Each category has a unique CategoryID.

```
CREATE TABLE Category (
    CategoryID INT PRIMARY KEY AUTO_INCREMENT,
    C_name VARCHAR(255),
    C_description TEXT,
    Address VARCHAR(255),
    Phone_Num VARCHAR(255)
);
```

Review Table

Stores reviews made by users on products. Each review has a unique ReviewID. Linked to Product via ProductID and User via UserID.

```
CREATE TABLE Review (
  ReviewID INT PRIMARY KEY AUTO_INCREMENT,
  ProductID INT,
  UserID INT,
  Rating INT,
  Comment TEXT,
  Review_Date DATETIME,
  FOREIGN KEY (ProductID) REFERENCES Product(ProductID),
  FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

Cart Table

Contains information about the user's cart. Each cart has a unique CartID. Linked to User via UserID.

```
CREATE TABLE Cart (
  CartID INT PRIMARY KEY AUTO_INCREMENT,
  UserID INT,
  FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

Cart_Item Table

Details of the items in a user's cart. Each cart item has a unique Cart_ItemID. Linked to Cart via CartID and Product via ProductID.

```
CREATE TABLE Cart_Item (
  Cart_ItemID INT PRIMARY KEY AUTO_INCREMENT,
  CartID INT,
  ProductID INT,
  Quantity INT,
```

```
    FOREIGN KEY (CartID) REFERENCES Cart(CartID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

5.SQL queries for common operations

Inserting a New User

```
INSERT INTO User (Name, Email, Password, Address, Phone_Num)
VALUES ('John Doe', 'johndoe@example.com', 'hashed_password', '123 Maple Street', '555-1234');
```

Adding a new Product:

```
INSERT INTO Product (P_name, P_description, Price, StockQuantity, CategoryID)
VALUES ('Gadget', 'Latest tech gadget', 299.99, 50, 1);
```

Creating a New Order

```
INSERT INTO Order (UserID, Order_Date, Shipping_Addr, Total_Price, Status)
VALUES (1, '2024-03-08 10:00:00', '123 Maple Street', 599.98, 'Processing');
```

Updating the Stock Quantity of a Product

```
UPDATE Product
SET StockQuantity = StockQuantity - 2
WHERE ProductID = 1;
```

Retrieving Information about a Specific Order

```
SELECT * FROM Order
WHERE OrderID = 1;
```

Fetching All Reviews for a Product

```
SELECT * FROM Review
WHERE ProductID = 1;
```

6.Integrity Constraints:

Ensure that each record within a table is unique and can be identified reliably.

```
ALTER TABLE User ADD CONSTRAINT PK_User PRIMARY KEY (UserID);
```

ALTER TABLE Admin ADD CONSTRAINT PK_Admin PRIMARY KEY (AdminID);

ALTER TABLE Order ADD CONSTRAINT PK_Order PRIMARY KEY (OrderID);

ALTER TABLE Order_Detail ADD CONSTRAINT PK_Order_Detail PRIMARY KEY (Order_DetailID);

ALTER TABLE Product ADD CONSTRAINT PK_Product PRIMARY KEY (ProductID);

ALTER TABLE Category ADD CONSTRAINT PK_Category PRIMARY KEY (CategoryID);

ALTER TABLE Review ADD CONSTRAINT PK_Review PRIMARY KEY (ReviewID);

ALTER TABLE Cart ADD CONSTRAINT PK_Cart PRIMARY KEY (CartID);

ALTER TABLE Cart_Item ADD CONSTRAINT PK_Cart_Item PRIMARY KEY (Cart_ItemID);

Foreign Key Constraints:

These ensure referential integrity, meaning that relationships between tables are consistent.

Order to User

ALTER TABLE Order ADD CONSTRAINT FK_Order_User FOREIGN KEY (UserID) REFERENCES User(UserID);

Order_Detail to Order

ALTER TABLE Order_Detail ADD CONSTRAINT FK_OrderDetail_Order FOREIGN KEY (OrderID) REFERENCES Order(OrderID);

Order_Detail to Product

ALTER TABLE Order_Detail ADD CONSTRAINT FK_OrderDetail_Product FOREIGN KEY (ProductID) REFERENCES Product(ProductID);

Review to Product

ALTER TABLE Review ADD CONSTRAINT FK_Review_Product FOREIGN KEY (ProductID) REFERENCES Product(ProductID);

Review to User

ALTER TABLE Review ADD CONSTRAINT FK_Review_User FOREIGN KEY (UserID) REFERENCES User(UserID);

Cart to User

ALTER TABLE Cart ADD CONSTRAINT FK_Cart_User FOREIGN KEY (UserID) REFERENCES User(UserID);

Cart_Item to Cart

ALTER TABLE Cart_Item ADD CONSTRAINT FK_CartItem_Cart FOREIGN KEY (CartID) REFERENCES Cart(CartID);

Cart_Item to Product

ALTER TABLE Cart_Item ADD CONSTRAINT FK_CartItem_Product FOREIGN KEY (ProductID) REFERENCES Product(ProductID);

Product to Category

ALTER TABLE Product ADD CONSTRAINT FK_Product_Category FOREIGN KEY (CategoryID) REFERENCES Category(CategoryID);

Unique Constraints:

Ensure that certain columns contain unique values, such as email addresses.

ALTER TABLE User ADD CONSTRAINT UN_User_Email UNIQUE (Email);

ALTER TABLE Admin ADD CONSTRAINT UN_Admin_Email UNIQUE (Email);

Check Constraints:

Enforce domain integrity by limiting the values that can be placed in a column.

A valid rating in Review (assuming 1 to 5 scale)

ALTER TABLE Review ADD CONSTRAINT CK_Review_Rating CHECK (Rating BETWEEN 1 AND 5);

Positive price in Product

ALTER TABLE Product ADD CONSTRAINT CK_Product_Price CHECK (Price > 0);

Non-negative stock quantity in Product

ALTER TABLE Product ADD CONSTRAINT CK_Product_StockQuantity CHECK (StockQuantity >= 0);

Total_Price in Order to be greater than zero

ALTER TABLE Order ADD CONSTRAINT CK_Order_TotalPrice CHECK (Total_Price > 0);

Cascading Actions for Referential Integrity:

Handle the updates and deletions between related tables appropriately.

Cascading delete for Order when a User is deleted

ALTER TABLE Order ADD CONSTRAINT FK_Order_User_Cascade FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE CASCADE;

Setting to NULL in Review when a User is deleted

ALTER TABLE Review ADD CONSTRAINT FK_Review_User_SetNull FOREIGN KEY (UserID) REFERENCES User(UserID) ON DELETE SET NULL;

Setting to NULL in Review when a Product is deleted

ALTER TABLE Review ADD CONSTRAINT FK_Review_Product_SetNull FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE SET NULL;

Cascading delete for Cart_Item when a Cart is deleted

ALTER TABLE Cart_Item ADD CONSTRAINT FK_CartItem_Cart_Cascade FOREIGN KEY (CartID) REFERENCES Cart(CartID) ON DELETE CASCADE;

Cascading delete for Cart_Item when a Product is deleted

ALTER TABLE Cart_Item ADD CONSTRAINT FK_CartItem_Product_Cascade FOREIGN KEY (ProductID) REFERENCES Product(ProductID) ON DELETE CASCADE;