## 1. Description of the Project

The goal of this project is to create a well-organized and reliable database system to support the main functions of an online shopping platform. This system is designed to handle important tasks like storing information about products, customers, orders, and payments.

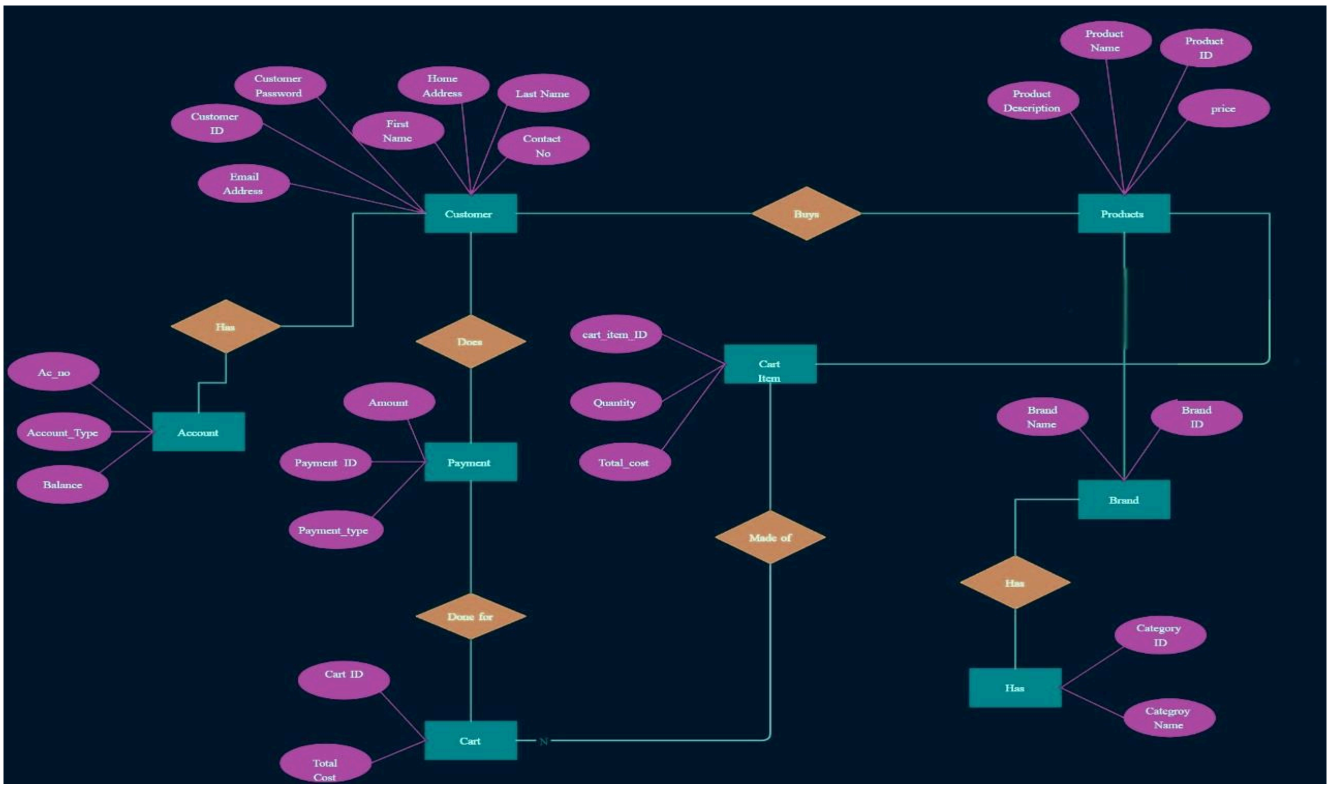
It helps the business manage its product inventory, making sure that levels are always up-to-date. It also keeps track of customer purchases, so the business knows what was ordered, when it was ordered, and by whom. On top of that, it allows the company to review sales data to understand customer behavior and make better business decisions.

The database uses **SQL** (Structured Query Language) to add, update, delete, and retrieve data. To keep the database well-structured and efficient, it uses key concepts from **relational database design**, such as:

- **Normalization**: Organizing data to reduce duplication and improve consistency.
- **Constraints**: Setting rules for the data to make sure it's accurate and valid.
- **Indexing**: Speeding up data search and access for better performance.

Together, these methods make sure the system runs smoothly, keeps data accurate, and handles large amounts of information without slowing down.

## 2. ER Diagram Creation



## 3. Description of ER Diagram

This **Entity-Relationship Diagram (ERD)** visually represents the data structure of an **Online Retail Shop** system. It includes key entities such as `Customer`, `Product`, `Payment`, `Cart`, `Account`, and their relationships. The design follows standard ERD notation using:

- **Rectangles** for entities
- **Ovals** for attributes
- **Diamonds** for relationships
- **Connecting lines** to represent associations and cardinality

### Entities & Attributes:

1. **Customer**
    - **Attributes**:
        - Customer_ID *(PK)*
        - First_Name
        - Last_Name
        - Email_Address
        - Contact_No
        - House_Address
        - Customer_Password
    - **Relationships**:
        - `Has` → Account
        - `Does` → Payment
        - `Buys` → Products

2. **Account**
   - **Attributes**:
     - Ac_no *(PK)*
     - Account_Type
     - Balance
   - **Relationships**:
     - `Has` ← Customer

3. **Payment**
   - **Attributes**:
     - Payment_ID *(PK)*
     - Amount
     - Payment_type
   - **Relationships**:
     - `Does` ← Customer
     - `Done For` → Cart

4. **Cart**
   - **Attributes**:
     - Cart_ID *(PK)*
     - Total_Cost
   - **Relationships**:
     - `Done For` ← Payment
     - `Made Of` → Cart_Item

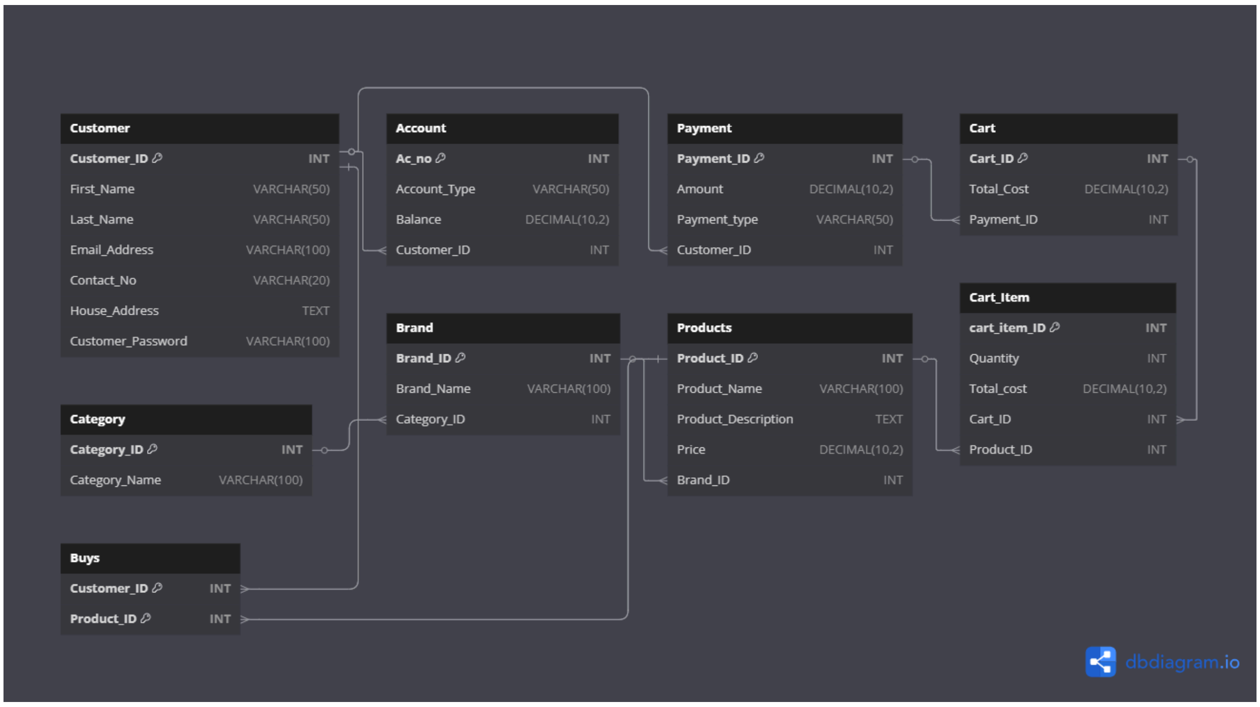
5. **Cart_Item**
    - **Attributes**:
        - cart_item_ID *(PK)*
        - Quantity
        - Total_cost
    - **Relationships**:
        - `Made Of` ← Cart
        - `Includes` → Products
6. **Products**
    - **Attributes**:
        - Product_ID *(PK)*
        - Product_Name
        - Product_Description
        - Price
    - **Relationships**:
        - `Buys` ← Customer
        - `Includes` ← Cart_Item
7. **Brand**
    - **Attributes**:
        - Brand_ID *(PK)*
        - Brand_Name
    - **Relationships**:
        - `Has` → Category

8. **Category**
   - **Attributes**:
     - Category_ID *(PK)*
     - Category_Name
   - **Relationships**:
     - `Has` ← Brand

## Relationships Summary:

- A **Customer** can have one **Account**
- A **Customer** can make multiple **Payments**
- A **Customer** can buy multiple **Products** (many-to-many via `Buys`)
- A **Payment** is for one **Cart**
- A **Cart** consists of multiple **Cart_Items**
- A **Cart_Item** is linked to one **Product**
- A **Product** belongs to one **Brand**
- A **Brand** belongs to one **Category**

## 4. Conversion of ER Diagram into Tables



SQL COMMANDS:-

```
CREATE TABLE Customer (

    Customer_ID INT PRIMARY KEY,

    First_Name VARCHAR(50),

    Last_Name VARCHAR(50),

    Email_Address VARCHAR(100),

    Contact_No VARCHAR(20),

    House_Address TEXT,

    Customer_Password VARCHAR(100)

);
```

```sql
CREATE TABLE Account (

    Ac_no INT PRIMARY KEY,

    Account_Type VARCHAR(50),

    Balance DECIMAL(10,2),

    Customer_ID INT,

    FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)

);


CREATE TABLE Payment (

    Payment_ID INT PRIMARY KEY,

    Amount DECIMAL(10,2),

    Payment_type VARCHAR(50),

    Customer_ID INT,

    FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)

);


CREATE TABLE Cart (

    Cart_ID INT PRIMARY KEY,

    Total_Cost DECIMAL(10,2),

    Payment_ID INT,

    FOREIGN KEY (Payment_ID) REFERENCES Payment(Payment_ID)

);


CREATE TABLE Category (

    Category_ID INT PRIMARY KEY,

    Category_Name VARCHAR(100)
```

```sql
);


CREATE TABLE Brand (
    Brand_ID INT PRIMARY KEY,
    Brand_Name VARCHAR(100),
    Category_ID INT,
    FOREIGN KEY (Category_ID) REFERENCES Category(Category_ID)
);


CREATE TABLE Products (
    Product_ID INT PRIMARY KEY,
    Product_Name VARCHAR(100),
    Product_Description TEXT,
    Price DECIMAL(10,2),
    Brand_ID INT,
    FOREIGN KEY (Brand_ID) REFERENCES Brand(Brand_ID)
);


CREATE TABLE Cart_Item (
    cart_item_ID INT PRIMARY KEY,
    Quantity INT,
    Total_cost DECIMAL(10,2),
    Cart_ID INT,
    Product_ID INT,
    FOREIGN KEY (Cart_ID) REFERENCES Cart(Cart_ID),
    FOREIGN KEY (Product_ID) REFERENCES Products(Product_ID)
```

```
);


CREATE TABLE Buys (
    Customer_ID INT,
    Product_ID INT,
    PRIMARY KEY (Customer_ID, Product_ID),
    FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID),
    FOREIGN KEY (Product_ID) REFERENCES Products(Product_ID)
);
```

## 5. Description of Tables

### i. Customer Table

Stores information about each registered customer.

| Column Name | Data Type | Description |
|---|---|---|
| `Customer_ID` | INT (PK) | Unique identifier for each customer. |
| `First_Name` | VARCHAR(50) | Customer’s first name. |
| `Last_Name` | VARCHAR(50) | Customer’s last name. |
| `Email_Address` | VARCHAR(100) | Customer’s email address. |
| `Contact_No` | VARCHAR(20) | Customer’s phone number. |

| | | |
|---|---|---|
| `House_Address` | TEXT | Physical address of the customer. |
| `Customer_Password` | VARCHAR(100) | Encrypted or hashed password for login. |

---

### ii. Account Table

Represents the customer's payment account or wallet within the system.

| Column Name | Data Type | Description |
|---|---|---|
| `Ac_no` | INT (PK) | Unique account number. |
| `Account_Type` | VARCHAR(50) | Type of account (e.g., savings, credit). |
| `Balance` | DECIMAL(10,2) | Current account balance. |
| `Customer_ID` | INT (FK) | Links to the respective customer. |

---

### iii. Payment Table

Stores payment transactions made by customers.

| Column Name | Data Type | Description |
|---|---|---|
| `Payment_ID` | INT (PK) | Unique identifier for each payment. |

| | | |
|---|---|---|
| `Amount` | DECIMAL(10,2) | Total amount paid. |
| `Payment_type` | VARCHAR(50) | Type of payment (e.g., credit card, PayPal). |
| `Customer_ID` | INT (FK) | The customer who made the payment. |

---

### iv. Cart Table

Tracks the shopping cart details per transaction.

| Column Name | Data Type | Description |
|---|---|---|
| `Cart_ID` | INT (PK) | Unique identifier for each cart. |
| `Total_Cost` | DECIMAL(10,2) | Total cost of all items in the cart. |
| `Payment_ID` | INT (FK) | Links to the payment made for this cart. |

### v. Category Table

Defines product categories for classification.

| Column Name | Data Type | Description |
|---|---|---|
| `Category_ID` | INT (PK) | Unique ID for each product category. |
| `Category_Name` | VARCHAR(100) | Name of the product category. |

---

### vi. Brand Table

Captures brands under each product category.

| Column Name | Data Type | Description |
|---|---|---|
| `Brand_ID` | INT (PK) | Unique identifier for each brand. |
| `Brand_Name` | VARCHAR(100) | Brand name. |
| `Category_ID` | INT (FK) | The category this brand belongs to. |

---

### vii. Products Table

Holds details of all products available in the store.

| Column Name | Data Type | Description |
|---|---|---|
| `Product_ID` | INT (PK) | Unique identifier for the product. |

| | | |
|---|---|---|
| `Product_Name` | VARCHAR(100) | Name of the product. |
| `Product_Description` | TEXT | Detailed description of the product. |
| `Price` | DECIMAL(10,2) | Product price. |
| `Brand_ID` | INT (FK) | Associated brand of the product. |

---

### vii. Cart_Item Table

Intermediate table linking products to carts (many-to-many relationship).

| Column Name | Data Type | Description |
|---|---|---|
| `cart_item_ID` | INT (PK) | Unique ID for each cart item. |
| `Quantity` | INT | Number of units of the product. |
| `Total_cost` | DECIMAL(10,2) | Cost of this cart item (quantity * price). |
| `Cart_ID` | INT (FK) | Cart in which the item exists. |
| `Product_ID` | INT (FK) | Product that has been added. |

---

### xi. Buys Table

A junction table to represent the many-to-many relationship between Customers and Products they purchase.

| Column Name | Data Type | Description |
|---|---|---|
| `Customer_ID` | INT (PK, FK) | Customer who purchased. |
| `Product_ID` | INT (PK, FK) | Product that was bought. |

## 6. Normalization of Tables (Up to 3NF)

### First Normal Form (1NF):

- All columns contain atomic (single) values.
- Each table has a primary key.
- No repeating groups.

All tables follow 1NF.

---

### Second Normal Form (2NF):

- Already in 1NF.
- All non-key attributes fully depend on the whole primary key.
- No partial dependencies.

All tables are in 2NF.
Note: The `Buys` table has a composite key (`Customer_ID`, `Product_ID`) but no extra attributes, so it meets 2NF.

---

### Third Normal Form (3NF):

- Already in 2NF.
- No transitive dependencies (non-key attribute depending on another non-key attribute).

All tables are in 3NF.
Each attribute depends only on the primary key of its table.

***The database is fully normalized up to Third Normal Form (3NF).***
***It eliminates redundancy, maintains data integrity, and ensures efficient organization.***

## 7. Creation of Data in the Tables

To insert data into the Online Retail Shop database, we use the `INSERT INTO` SQL statement. Below is the complete set of sample data creation statements for all tables. This ensures that data is consistent and referential integrity is maintained.

---

### i. Customer Table

INSERT INTO Customer VALUES

(1, 'John', 'Doe', 'john@example.com', '1234567890', '123 Main St', 'password123'),

(2, 'Jane', 'Smith', 'jane@example.com', '9876543210', '456 Elm St', 'securepass456');

---

### ii. Account Table

INSERT INTO Account VALUES

(101, 'Savings', 5000.00, 1),

(102, 'Current', 12000.50, 2);

---

### iii. Payment Table

INSERT INTO Payment VALUES

(201, 250.00, 'Credit Card', 1),

(202, 99.99, 'PayPal', 2);

---

### iv. Cart Table

INSERT INTO Cart VALUES

(301, 349.99, 201),

(302, 99.99, 202);

---

### v. Category Table

INSERT INTO Category VALUES

(1, 'Electronics'),

(2, 'Clothing');

---

### vi. Brand Table

INSERT INTO Brand VALUES

(1, 'Samsung', 1),

(2, 'Nike', 2);

---

### vii. Products Table

INSERT INTO Products VALUES

(1001, 'Smartphone', 'Latest model with 128GB storage', 699.99, 1),

(1002, 'Running Shoes', 'Comfortable and durable', 89.99, 2);

---

### viii. Cart_Item Table

INSERT INTO Cart_Item VALUES

(401, 1, 699.99, 301, 1001),

(402, 1, 89.99, 302, 1002);

---

### ix. Buys Table

INSERT INTO Buys VALUES

(1, 1001),

(2, 1002);

# 8. Write SQL Queries

## 1. Subqueries

a. Find customers who made payments above the average payment amount

```
SELECT First_Name, Last_Name
 FROM Customer
 WHERE Customer_ID IN (
       SELECT Customer_ID
       FROM Payment
       WHERE Amount > (SELECT AVG(Amount) FROM Payment)
 );
```

b. List products that are in any cart

```
SELECT Product_Name
 FROM Products
 WHERE Product_ID IN (
       SELECT Product_ID FROM Cart_Item
 );
```

## 2. Aggregate Functions

a. Total revenue generated from all products

```
SELECT SUM(Total_cost) AS Total_Revenue
 FROM Cart_Item;
```

b. Count the number of customers

```
SELECT COUNT(*) AS Total_Customers
 FROM Customer;
```

c. Find the average price of all products

```
SELECT AVG(Price) AS Average_Price
 FROM Products;
```

## 3. Joins

a. Get customer names with their payment details

```
SELECT c.First_Name, c.Last_Name, p.Amount, p.Payment_type
 FROM Customer c
 JOIN Payment p ON c.Customer_ID = p.Customer_ID;
```

b. Show cart items with product names and their quantities

```
SELECT ci.cart_item_ID, p.Product_Name, ci.Quantity, ci.Total_cost
 FROM Cart_Item ci
 JOIN Products p ON ci.Product_ID = p.Product_ID;
```

c. List products with their brand and category names

```
SELECT p.Product_Name, b.Brand_Name, c.Category_Name
 FROM Products p
 JOIN Brand b ON p.Brand_ID = b.Brand_ID
 JOIN Category c ON b.Category_ID = c.Category_ID;
```

d. List all products bought by each customer

```
SELECT cu.First_Name, cu.Last_Name, pr.Product_Name
 FROM Buys b
 JOIN Customer cu ON b.Customer_ID = cu.Customer_ID
 JOIN Products pr ON b.Product_ID = pr.Product_ID;
```

## 9. Creation of Views

1. **View for Customer Payment Information:** This view combines customer details with their payment information, showing the customer's name and their payment amount and type.

```
CREATE VIEW CustomerPaymentInfo AS
SELECT c.First_Name, c.Last_Name, p.Amount, p.Payment_type
FROM Customer c
JOIN Payment p ON c.Customer_ID = p.Customer_ID;
```

2. **View for Cart Details with Product Information:** This view lists all the cart items along with the product names, quantities, and total cost for each item in the cart.

```
CREATE VIEW CartProductDetails AS
SELECT ci.cart_item_ID, p.Product_Name, ci.Quantity, ci.Total_cost
FROM Cart_Item ci
JOIN Products p ON ci.Product_ID = p.Product_ID;
```

3. **View for Products with Brand and Category Information:** This view combines product details with their associated brand and category names.

```
CREATE VIEW ProductBrandCategory AS
SELECT p.Product_Name, b.Brand_Name, c.Category_Name
FROM Products p
JOIN Brand b ON p.Brand_ID = b.Brand_ID
JOIN Category c ON b.Category_ID = c.Category_ID;
```

4. **View for Customer's Purchased Products:** This view lists the products bought by each customer, showing the customer's name and the product they purchased.

```
CREATE VIEW CustomerPurchasedProducts AS
SELECT cu.First_Name, cu.Last_Name, pr.Product_Name
FROM Buys b
JOIN Customer cu ON b.Customer_ID = cu.Customer_ID
JOIN Products pr ON b.Product_ID = pr.Product_ID;
```

5. **View for Total Revenue from Cart Items:** This view calculates the total revenue generated from the cart items, summing up the total cost of all items in the carts.

```
CREATE VIEW TotalRevenue AS
SELECT SUM(Total_cost) AS Total_Revenue
FROM Cart_Item;
```

6. **View for Average Product Price:** This view calculates the average price of all products in the store.

```
CREATE VIEW AverageProductPrice AS
SELECT AVG(Price) AS Average_Price
FROM Products;
```

7. **View for Customers Who Made Payments Above the Average:** This view lists the customers who have made payments above the average payment amount.

```
CREATE VIEW CustomersAboveAvgPayment AS
SELECT c.First_Name, c.Last_Name
FROM Customer c
WHERE c.Customer_ID IN (
   SELECT Customer_ID
   FROM Payment
   WHERE Amount > (SELECT AVG(Amount) FROM Payment)
);
```