# Inventory And Warehouse Management Database System

**Name : Dev Singh Shekhawat Roll No. : 2310110099**

**Name : Neeraj Yadav Roll No. : 2310110199**

**OBJECTIVES :**  
To develop a simple and efficient inventory management system tailored for small businesses, such as Instagram boutique sellers, enabling them to track products, manage stock levels, handle orders, and streamline warehouse operations with minimal technical effort.

# Creating The Database Schema :

CREATE DATABASE ProjectInventoryAndWarehouseManagement; USE ProjectInventoryAndWarehouseManagement;

#### -- CUSTOMER

CREATE TABLE CUSTOMER (

CustomerID INT AUTO\_INCREMENT PRIMARY KEY, Name VARCHAR(255) NOT NULL,

ContactInfo VARCHAR(255) NOT NULL, Address VARCHAR(255) NOT NULL

);

#### -- SUPPLIER

CREATE TABLE SUPPLIER (

SupplierID INT AUTO\_INCREMENT PRIMARY KEY, Name VARCHAR(255) NOT NULL,

ContactInfo VARCHAR(255) NOT NULL

);

#### -- PRODUCT

CREATE TABLE PRODUCT (

ProductID INT AUTO\_INCREMENT PRIMARY KEY, Name VARCHAR(255) NOT NULL,

Description TEXT, SupplierID INT,

FOREIGN KEY (SupplierID) REFERENCES SUPPLIER(SupplierID) ON DELETE CASCADE

);

#### -- WAREHOUSE

CREATE TABLE WAREHOUSE (

WarehouseID INT AUTO\_INCREMENT PRIMARY KEY, Location VARCHAR(255) NOT NULL,

Capacity INT NOT NULL

);

#### -- SHIPMENT

CREATE TABLE SHIPMENT (

ShipmentID INT AUTO\_INCREMENT PRIMARY KEY,

ShipmentDate DATE NOT NULL, WarehouseID INT,

FOREIGN KEY (WarehouseID) REFERENCES WAREHOUSE(WarehouseID) ON DELETE CASCADE

);

#### -- ORDERS

CREATE TABLE ORDERS (

OrderID INT AUTO\_INCREMENT PRIMARY KEY,

OrderDate DATE NOT NULL, CustomerID INT,

ShipmentID INT,

FOREIGN KEY (CustomerID) REFERENCES CUSTOMER(CustomerID) ON DELETE CASCADE, FOREIGN KEY (ShipmentID) REFERENCES SHIPMENT(ShipmentID) ON DELETE CASCADE

);

#### -- ORDER\_PRODUCT

CREATE TABLE ORDER\_PRODUCT (

OrderProductID INT AUTO\_INCREMENT PRIMARY KEY,

OrderID INT, ProductID INT,

QuantityOrdered INT NOT NULL,

FOREIGN KEY (OrderID) REFERENCES ORDERS(OrderID) ON DELETE CASCADE,

FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID) ON DELETE CASCADE

);

#### -- WAREHOUSE\_INVENTORY

CREATE TABLE WAREHOUSE\_INVENTORY (

WarehouseInventoryID INT AUTO\_INCREMENT PRIMARY KEY, WarehouseID INT,

ProductID INT,

Quantity INT NOT NULL DEFAULT 0,

FOREIGN KEY (WarehouseID) REFERENCES WAREHOUSE(WarehouseID) ON DELETE CASCADE,

FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID) ON DELETE CASCADE

);

#### -- INVENTORY\_MOVEMENT

CREATE TABLE INVENTORY\_MOVEMENT (

MovementID INT AUTO\_INCREMENT PRIMARY KEY,

ProductID INT, WarehouseFromID INT, WarehouseToID INT, OrderID INT,

ShipmentID INT, QuantityMoved INT NOT NULL,

MovementDate DATE NOT NULL,

MovementType ENUM('Transfer', 'Restock', 'Sale') NOT NULL,

FOREIGN KEY (ProductID) REFERENCES PRODUCT(ProductID) ON DELETE CASCADE,

FOREIGN KEY (WarehouseFromID) REFERENCES WAREHOUSE(WarehouseID) ON DELETE CASCADE,

FOREIGN KEY (WarehouseToID) REFERENCES WAREHOUSE(WarehouseID) ON DELETE CASCADE,

FOREIGN KEY (OrderID) REFERENCES ORDERS(OrderID) ON DELETE CASCADE,

FOREIGN KEY (ShipmentID) REFERENCES SHIPMENT(ShipmentID) ON DELETE CASCADE

);

-- USERS TABLE

CREATE TABLE USERS (

UserID INT AUTO\_INCREMENT PRIMARY KEY,

Username VARCHAR(255) NOT NULL UNIQUE,

PasswordHash VARCHAR(255) NOT NULL,

Role ENUM('Admin', 'Manager') NOT NULL

);

-- USER-WAREHOUSE MAPPING (only for Managers)

CREATE TABLE USER\_WAREHOUSE (

UserID INT,

WarehouseID INT,

PRIMARY KEY (UserID, WarehouseID),

FOREIGN KEY (UserID) REFERENCES USERS(UserID) ON DELETE CASCADE,

FOREIGN KEY (WarehouseID) REFERENCES WAREHOUSE(WarehouseID) ON DELETE CASCADE

);

-- Triggers

DELIMITER $$

CREATE TRIGGER after\_order\_product\_insert

AFTER INSERT ON ORDER\_PRODUCT

FOR EACH ROW

BEGIN

DECLARE order\_warehouse\_id INT;

-- Get the warehouse fulfilling this order via SHIPMENT

SELECT s.WarehouseID INTO order\_warehouse\_id

FROM ORDERS o

JOIN SHIPMENT s ON o.ShipmentID = s.ShipmentID

WHERE o.OrderID = NEW.OrderID;

-- Decrease the quantity in the warehouse inventory

UPDATE WAREHOUSE\_INVENTORY

SET Quantity = Quantity - NEW.QuantityOrdered

WHERE WarehouseID = order\_warehouse\_id

AND ProductID = NEW.ProductID;

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER after\_inventory\_movement\_insert

AFTER INSERT ON INVENTORY\_MOVEMENT

FOR EACH ROW

BEGIN

-- Decrease from source warehouse

IF NEW.WarehouseFromID IS NOT NULL THEN

UPDATE WAREHOUSE\_INVENTORY

SET Quantity = Quantity - NEW.QuantityMoved

WHERE WarehouseID = NEW.WarehouseFromID

AND ProductID = NEW.ProductID;

END IF;

-- Increase at destination warehouse

IF NEW.WarehouseToID IS NOT NULL THEN

INSERT INTO WAREHOUSE\_INVENTORY (WarehouseID, ProductID, Quantity)

VALUES (NEW.WarehouseToID, NEW.ProductID, NEW.QuantityMoved)

ON DUPLICATE KEY UPDATE Quantity = Quantity + NEW.QuantityMoved;

END IF;

END $$

DELIMITER ;

DELIMITER $$

CREATE TRIGGER before\_delete\_product

BEFORE DELETE ON PRODUCT

FOR EACH ROW

BEGIN

IF EXISTS (

SELECT 1 FROM WAREHOUSE\_INVENTORY

WHERE ProductID = OLD.ProductID AND Quantity > 0

) THEN

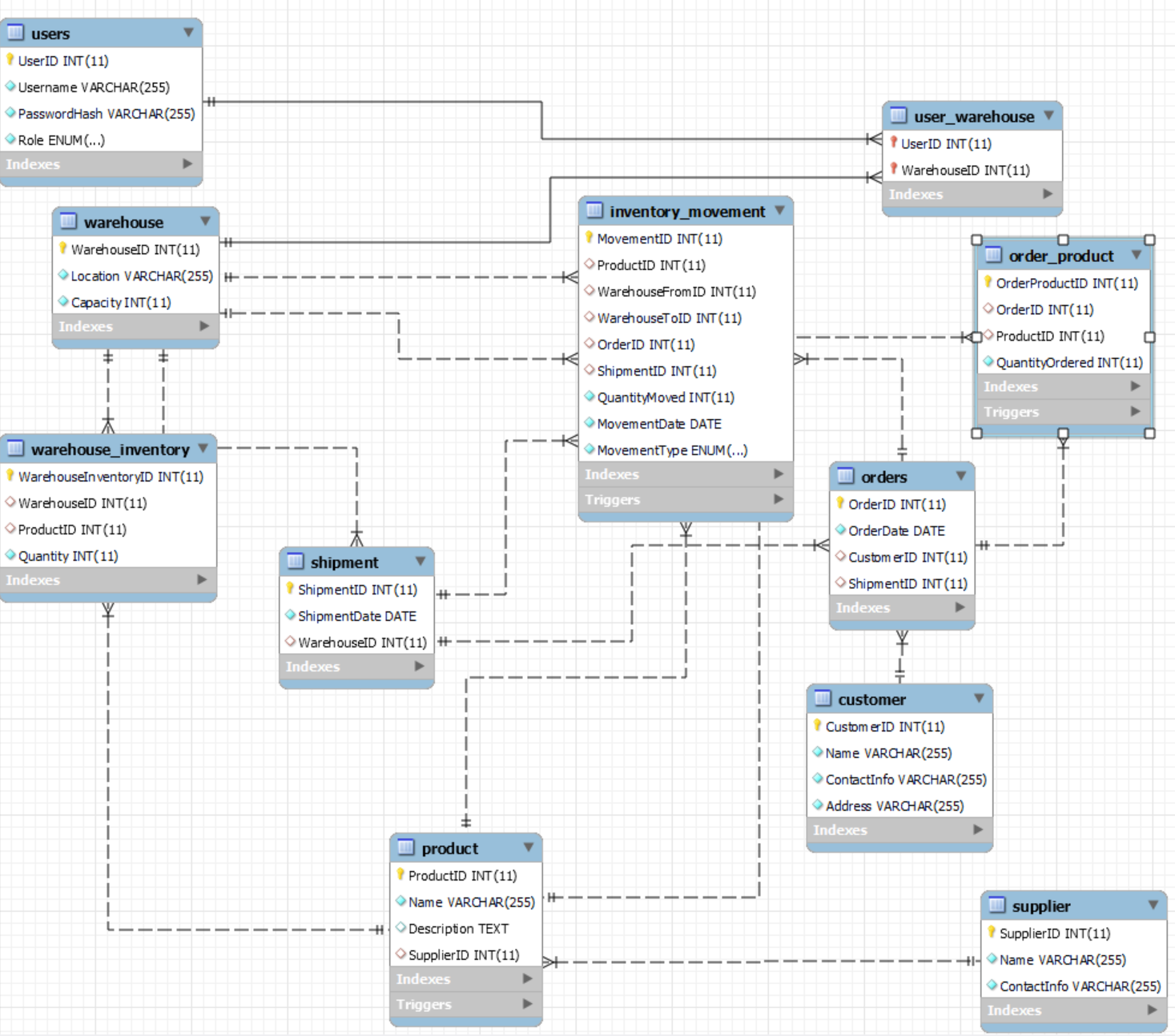
SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot delete product with existing inventory.';

END IF;

END $$

DELIMITER ;



# Inserting sample data :

#### -- CUSTOMER

INSERT INTO CUSTOMER (Name, ContactInfo, Address) VALUES

('Amit Sharma', ['amit.sharma@gmail.com',](mailto:%27amit.sharma@gmail.com) 'Sector 15, Gurugram, Haryana'),

('Priya Iyer', ['priya.iyer@gmail.com',](mailto:%27priya.iyer@gmail.com) 'JP Nagar, Bangalore, Karnataka'),

('Rahul Singh', ['rahul.singh@gmail.com',](mailto:%27rahul.singh@gmail.com) 'Powai, Mumbai, Maharashtra'),

('Neha Verma', ['neha.verma@gmail.com',](mailto:%27neha.verma@gmail.com) 'Salt Lake, Kolkata, West Bengal'), ('Ravi Patel', ['ravi.patel@gmail.com',](mailto:%27ravi.patel@gmail.com) 'Satellite, Ahmedabad, Gujarat');

#### -- SUPPLIER

INSERT INTO SUPPLIER (Name, ContactInfo) VALUES

('Bharat Electronics Ltd.', ['contact@belindia.com'),](mailto:%27contact@belindia.com) ('Tata Advanced Systems', ['support@tataas.com'),](mailto:%27support@tataas.com) ('Havells India Ltd.', ['info@havells.com'),](mailto:%27info@havells.com)

('Godrej Consumer Products', ['care@godrej.com'),](mailto:%27care@godrej.com)

('Samsung India Electronics', ['service@samsungindia.com');](mailto:%27service@samsungindia.com)

#### -- PRODUCT

INSERT INTO PRODUCT (Name, Description, SupplierID) VALUES

('Smartphone Z1', '5G-enabled smartphone with AI camera', 5), ('LED TV 55"', '55-inch 4K Ultra HD Smart TV', 3),

('Refrigerator DX200', 'Double-door frost-free refrigerator', 4),

('Laptop Elite 15', 'High-performance laptop for professionals', 2), ('Air Conditioner Pro', '1.5 Ton Inverter AC with WiFi control', 1);

#### -- WAREHOUSE

INSERT INTO WAREHOUSE (Location, Capacity) VALUES

('Delhi', 5000),

('Mumbai', 4500),

('Bangalore', 4000),

('Chennai', 3500),

('Hyderabad', 3000);

#### -- SHIPMENT

INSERT INTO SHIPMENT (ShipmentDate, WarehouseID) VALUES

('2024-03-10', 1),

('2024-03-15', 2),

('2024-03-20', 3),

('2024-04-01', 4),

('2024-04-05', 5);

#### -- ORDERS

INSERT INTO ORDERS (OrderDate, CustomerID, ShipmentID) VALUES

('2024-03-11', 1, 1),

('2024-03-16', 2, 2),

('2024-03-21', 3, 3),

('2024-04-02', 4, 4),

('2024-04-06', 5, 5),

('2024-04-08', 1, 1),

('2024-04-10', 2, 2);

#### -- ORDER\_PRODUCT

INSERT INTO ORDER\_PRODUCT (OrderID, ProductID, QuantityOrdered) VALUES

(1, 1, 2),

(1, 2, 1),

(2, 3, 3),

(3, 4, 2),

(4, 5, 5),

(5, 1, 3),

(6, 2, 2),

(6, 3, 1),

(7, 4, 4),

(7, 5, 2);

#### -- WAREHOUSE\_INVENTORY

INSERT INTO WAREHOUSE\_INVENTORY (WarehouseID, ProductID, Quantity) VALUES

(1, 1, 50),

(1, 2, 30),

(2, 3, 20),

(3, 4, 15),

(4, 5, 25),

(5, 1, 10),

(2, 2, 40);

#### -- INVENTORY\_MOVEMENT

INSERT INTO INVENTORY\_MOVEMENT (ProductID, WarehouseFromID, WarehouseToID, OrderID, ShipmentID, QuantityMoved, MovementDate, MovementType)

#### VALUES

(1, 1, 2, 1, 1, 5, '2024-03-11', 'Transfer'),

(2, 1, 3, 1, 1, 3, '2024-03-11', 'Sale'),

(3, 2, 2, 2, 2, 7, '2024-03-16', 'Sale'),

(4, 3, 4, 3, 3, 4, '2024-03-21', 'Transfer'),

(5, 4, 5, 4, 4, 10, '2024-04-02', 'Restock'),

(1, 5, 1, 5, 5, 2, '2024-04-06', 'Transfer');

-- Insert sample warehouses

INSERT INTO WAREHOUSE (Location, Capacity) VALUES

('New York', 1000),

('Los Angeles', 800),

('Chicago', 1200);

-- Insert sample users

INSERT INTO USERS (Username, PasswordHash, Role) VALUES

('admin1', 'hash\_admin1', 'Admin'),

('admin2', 'hash\_admin2', 'Admin'),

('manager1', 'hash\_manager1', 'Manager'),

('manager2', 'hash\_manager2', 'Manager'),

('manager3', 'hash\_manager3', 'Manager');

-- Map managers to warehouses

INSERT INTO USER\_WAREHOUSE (UserID, WarehouseID) VALUES

(3, 1),

(3, 2),

(4, 2),

(5, 1),

(5, 3);

**Triggers**

DELIMITER //

CREATE TRIGGER set\_movement\_time

BEFORE INSERT ON INVENTORY\_MOVEMENT

FOR EACH ROW

BEGIN

SET NEW.MovementDate = CURTIME();

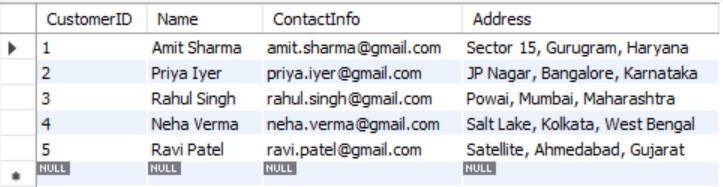
END;

//

DELIMITER ;

# Runned Queries With Their Outputs :

-- 1. Listing All Customers SELECT \* FROM CUSTOMER;



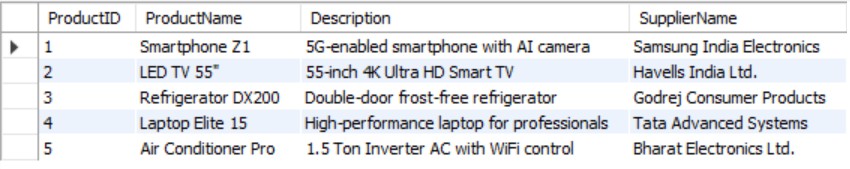
-- 2. Listing Products with Their Supplier Names SELECT

p.ProductID,

p.Name AS ProductName, p.Description,

s.Name AS SupplierName FROM PRODUCT p

JOIN SUPPLIER s ON p.SupplierID = s.SupplierID;



-- 3. Listing Orders with Customer and Shipment Details SELECT

o.OrderID, o.OrderDate,

c.Name AS CustomerName, sh.ShipmentDate

FROM ORDERS o

JOIN CUSTOMER c ON o.CustomerID = c.CustomerID

JOIN SHIPMENT sh ON o.ShipmentID = sh.ShipmentID;



-- 4. Listing Order Details: Products and Quantities SELECT

op.OrderID,

p.Name AS ProductName, op.QuantityOrdered

FROM ORDER\_PRODUCT op

JOIN PRODUCT p ON op.ProductID = p.ProductID;



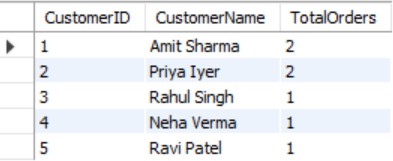
-- 5. Counting the Number of Orders per Customer SELECT

c.CustomerID,

c.Name AS CustomerName,

COUNT(o.OrderID) AS TotalOrders FROM CUSTOMER c

JOIN ORDERS o ON c.CustomerID = o.CustomerID GROUP BY c.CustomerID, c.Name;



-- 6. Total Quantity Ordered for Each Product SELECT

p.ProductID,

p.Name AS ProductName,

SUM(op.QuantityOrdered) AS TotalQuantityOrdered FROM PRODUCT p

JOIN ORDER\_PRODUCT op ON p.ProductID = op.ProductID GROUP BY p.ProductID, p.Name;



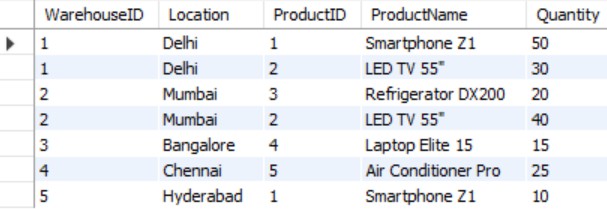
-- 7. Inventory Summary by Warehouse and Product SELECT

w.WarehouseID, w.Location, p.ProductID,

p.Name AS ProductName, wi.Quantity

FROM WAREHOUSE\_INVENTORY wi

JOIN WAREHOUSE w ON wi.WarehouseID = w.WarehouseID JOIN PRODUCT p ON wi.ProductID = p.ProductID;



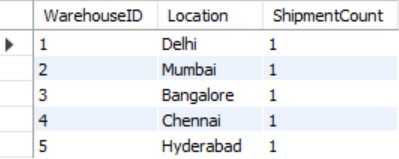
-- 8. Total Shipments per Warehouse SELECT

w.WarehouseID, w.Location,

COUNT(sh.ShipmentID) AS ShipmentCount FROM WAREHOUSE w

JOIN SHIPMENT sh ON w.WarehouseID = sh.WarehouseID

GROUP BY w.WarehouseID, w.Location;



-- 9. Inventory Movement details with Customer and Product Information SELECT

im.MovementID, im.MovementDate, im.MovementType,

p.Name AS ProductName, c.Name AS CustomerName, im.QuantityMoved

FROM INVENTORY\_MOVEMENT im

JOIN PRODUCT p ON im.ProductID = p.ProductID JOIN ORDERS o ON im.OrderID = o.OrderID

JOIN CUSTOMER c ON o.CustomerID = c.CustomerID;

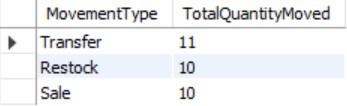


-- 10. Total Quantity Moved per Movement Type SELECT

im.MovementType,

SUM(im.QuantityMoved) AS TotalQuantityMoved FROM INVENTORY\_MOVEMENT im

GROUP BY im.MovementType;



## Data Constraints :

Our Database Has Primary Key Constraints, Auto-Increment Constraints, Not Null Constraints, Foreign Key Constraints, Cascade Delete Constraints, Default Value Constraints.

## Analysis of Functional Dependencies :

### CUSTOMER:

* + CustomerID → Name, ContactInfo, Address

### SUPPLIER:

* + SupplierID → Name, ContactInfo

### PRODUCT:

* + ProductID → Name, Description, SupplierID

### WAREHOUSE:

* + WarehouseID → Location, Capacity

### SHIPMENT:

* + ShipmentID → ShipmentDate, WarehouseID

### ORDERS:

* + OrderID → OrderDate, CustomerID, ShipmentID

### ORDER\_PRODUCT:

* + OrderProductID → OrderID, ProductID, QuantityOrdered
  + (OrderID, ProductID) → QuantityOrdered

### WAREHOUSE\_INVENTORY:

* + WarehouseInventoryID → WarehouseID, ProductID, Quantity
  + (WarehouseID, ProductID) → Quantity

### INVENTORY\_MOVEMENT:

* + MovementID → ProductID, WarehouseFromID, WarehouseToID, OrderID, ShipmentID, QuantityMoved, MovementDate, MovementType

## Analysis of highest normal form for the tables :

### CUSTOMER

* + **Attributes:** CustomerID (PK), Name, ContactInfo, Address
  + **Analysis:**
    - **1NF:** All attributes are atomic.
    - **2NF:** All non-key attributes are fully dependent on CustomerID.
    - **3NF:** No transitive dependencies exist.
    - **BCNF:** Every determinant is a candidate key.
  + **Highest Normal Form: BCNF**

### SUPPLIER

* + **Attributes:** SupplierID (PK), Name, ContactInfo
  + **Analysis:**
    - **1NF:** All attributes are atomic.
    - **2NF:** All non-key attributes are fully dependent on SupplierID.
    - **3NF:** No transitive dependencies exist.
    - **BCNF:** Every determinant is a candidate key.
  + **Highest Normal Form: BCNF**

### PRODUCT

* + **Attributes:** ProductID (PK), Name, Description, SupplierID
  + **Analysis:**
    - **1NF:** Atomic attributes.
    - **2NF:** Every non-key attribute is fully dependent on ProductID.
    - **3NF:** No transitive dependencies (SupplierID is a foreign key, but it does not lead to a transitive dependency within PRODUCT).
    - **BCNF:** Every determinant is a candidate key.
  + **Highest Normal Form: BCNF**

### WAREHOUSE

* + **Attributes:** WarehouseID (PK), Location, Capacity
  + **Analysis:**
    - **1NF, 2NF, 3NF:** All attributes are atomic and fully dependent on the primary key.
    - **BCNF:** Satisfied since every determinant is the candidate key.
  + **Highest Normal Form: BCNF**

### SHIPMENT

* + **Attributes:** ShipmentID (PK), ShipmentDate, WarehouseID
  + **Analysis:**
    - **1NF, 2NF, 3NF:** Each attribute is atomic and fully dependent on ShipmentID.
    - **BCNF:** Satisfied since ShipmentID is the sole candidate key.
  + **Highest Normal Form: BCNF**

### ORDERS

* + **Attributes:** OrderID (PK), OrderDate, CustomerID, ShipmentID
  + **Analysis:**
    - **1NF, 2NF, 3NF:** All non-key attributes are fully dependent on the primary key OrderID, and no transitive dependencies exist.
    - **BCNF:** Every determinant is a candidate key.
  + **Highest Normal Form: BCNF**

### ORDER\_PRODUCT

* + **Attributes:** OrderProductID (PK), OrderID, ProductID, QuantityOrdered
  + **Analysis:**
    - **1NF:** Atomic values.
    - **2NF:** All attributes are fully dependent on the primary key OrderProductID.
    - **Alternate Candidate Key:** (OrderID, ProductID) also uniquely determines QuantityOrdered.
    - **3NF/BCNF:** Both dependencies (using OrderProductID or the composite candidate key) satisfy BCNF as every determinant is a candidate key.
  + **Highest Normal Form: BCNF**

### WAREHOUSE\_INVENTORY

* + **Attributes:** WarehouseInventoryID (PK), WarehouseID, ProductID, Quantity
  + **Analysis:**
    - **1NF:** Atomic values.
    - **2NF:** All attributes are fully dependent on WarehouseInventoryID.
    - **Alternate Candidate Key:** (WarehouseID, ProductID) uniquely determines Quantity.
    - **3NF/BCNF:** The dependency from the candidate key (WarehouseID, ProductID) meets BCNF requirements.
  + **Highest Normal Form: BCNF**

### INVENTORY\_MOVEMENT

* + **Attributes:** MovementID (PK), ProductID, WarehouseFromID, WarehouseToID, OrderID, ShipmentID, QuantityMoved, MovementDate, MovementType
  + **Analysis:**
    - **1NF:** All attributes are atomic.
    - **2NF:** Every attribute is fully dependent on MovementID.
    - **3NF:** There are no transitive dependencies.
    - **BCNF:** Every determinant is a candidate key.
  + **Highest Normal Form: BCNF**

1. **Users**

**• Attributes:** UserID (PK), Username, PasswordHash, Role

**• Analysis:**

**o 1NF:** All attributes contain atomic values**.**

**o 2NF:** All attributes are fully dependent on the primary key UserID.

**o 3NF**: Username is unique and determines UserID, creating a transitive dependency (Username → UserID → other attributes), but Username is not a superkey.

**o BCNF: Violated due to Username →** UserID and Username not being a superkey.

**• Highest Normal Form: 3NF**

1. **User\_Warehouse**

**• Attributes:** UserID (PK-part), WarehouseID (PK-part)

**• Analysis:**

**o 1NF:** All values are atomic**.**

**o 2NF:** Composite key (UserID, WarehouseID) fully determines all data.

**o 3NF:** No transitive dependencies; foreign keys do not introduce anomalies.

**o BCNF:** All functional dependencies have a superkey as the determinant.

**• Highest Normal Form: BCNF**

**Frontend Explanation (HTML + CSS + JS)**

The frontend of the project is designed to provide a user-friendly interface for two types of users: Managers and Admins. Each user role has access to different functionalities, and this is reflected in their respective dashboards.

**Login System**

* **Users log in through a common login page.**
* **Based on credentials and roles fetched from the backend, users are redirected to:**
  + Admin Dashboard
  + Manager Dashboard

**Role-Based Dashboards**

* **The Admin Dashboard includes:**
  + User management
  + Supplier and customer data
  + Full inventory visibility
* **The Manager Dashboard includes:**
  + Order management
  + Stock levels and shipment tracking
  + Limited access to user controls

**Navigation and Interactivity**

* **Each dashboard has navigation options leading to different tasks.**
* **Selecting an option dynamically loads the corresponding HTML file (webpage), allowing the user to:**
  + Add, update, or delete records
  + View reports or logs
  + Track inventory movements
* **Pages are linked via buttons and navigation bars, making it easy to switch between modules.**

**Backend Integration (Flask)**

* **The frontend is powered by Flask on the backend.**
* **Flask handles:**
  + Routing between pages (@app.route)
  + Form submissions from HTML pages
  + Database interactions using SQLAlchemy or MySQL connectors
* **Dynamic data (like product lists, inventory updates, etc.) is fetched via Flask and rendered on the frontend as needed.**