Inventory management system

1)Entities and Attributes

1. Products

- ProductID (Primary Key)
- Name
- CategoryID (Foreign Key → Categories.CategoryID)
- StockLevel
- Price
- ReorderLevel (Threshold for low stock alerts)
- SupplierID (Foreign Key → Suppliers.SupplierID)

2. Orders

- OrderID (Primary Key)
- ProductID (Foreign Key → Products.ProductID)
- UserID (Foreign Key → Users.UserID, the one placing the order)
- Quantity
- OrderDate
- Status (Pending, Shipped, Delivered, Cancelled)

3. Suppliers

- SupplierID (Primary Key)
- Name
- ContactInfo
- Address

4. Users

- UserID (Primary Key)
- Username
- PasswordHash
- RoleID (Foreign Key → Roles.RoleID)

5. Roles

- RoleID (Primary Key)
- RoleName (BusinessOwner, Admin, WarehouseManager)

6. Sales

- SaleID (Primary Key)
- CustomerID (Foreign Key → Customers.CustomerID)
- ProductID (Foreign Key → Products.ProductID)
- SaleDate
- QuantitySold

- TotalPrice

7. Customers

- CustomerID (Primary Key)
- Name
- Email
- PhoneNumber
- Address

8. Inventory_Audit (To track stock changes)

- AuditID (Primary Key)
- ProductID (Foreign Key → Products.ProductID)
- ActionType (Stock Added, Stock Reduced, Damaged, etc.)
- QuantityChanged
- ChangeDate
- UserID (Foreign Key → Users.UserID, the person who made the change)

9. Warehouses

- WarehouseID (Primary Key)
- Location
- Capacity

10. Warehouse_Stock (Many-to-Many: Products stored in different warehouses)

- WarehouseID (Foreign Key → Warehouses.WarehouseID)
- ProductID (Foreign Key → Products.ProductID)
- StockQuantity

2)Relationships

1. Products → Orders (One-to-Many)
2. Orders → Users (Many-to-One, Business Owners place orders)
3. Orders → Suppliers (Many-to-One through Products)
4. Users → Roles (Many-to-One)
5. Sales → Customers (Many-to-One, Customers can have multiple purchases)
6. Sales → Products (Many-to-One, Products appear in multiple sales)
7. Inventory_Audit → Products (Many-to-One, tracking stock changes)
8. Warehouses → Products (Many-to-Many via Warehouse_Stock)

3) creating tables:

```
CREATE DATABASE InventoryManagement;
USE InventoryManagement;

-- 1. Categories Table
CREATE TABLE Categories (
    CategoryID INT PRIMARY KEY AUTO_INCREMENT,
    CategoryName VARCHAR(255) NOT NULL UNIQUE
);

-- 2. Products Table
CREATE TABLE Products (
    ProductID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    CategoryID INT,
    StockLevel INT DEFAULT 0,
    Price DECIMAL(10,2) NOT NULL,
```

```
    ReorderLevel INT DEFAULT 10,
    SupplierID INT,
    FOREIGN KEY (CategoryID) REFERENCES
Categories(CategoryID) ON DELETE SET NULL,
    FOREIGN KEY (SupplierID) REFERENCES
Suppliers(SupplierID) ON DELETE SET NULL
);

-- 3. Suppliers Table
CREATE TABLE Suppliers (
    SupplierID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    ContactInfo VARCHAR(255),
    Address TEXT
);

-- 4. Users Table
CREATE TABLE Users (
    UserID INT PRIMARY KEY AUTO_INCREMENT,
```

```
    Username VARCHAR(100) NOT NULL UNIQUE,
    PasswordHash VARCHAR(255) NOT NULL,
    RoleID INT,
    FOREIGN KEY (RoleID) REFERENCES Roles(RoleID) ON
DELETE SET NULL
);

-- 5. Roles Table
CREATE TABLE Roles (
    RoleID INT PRIMARY KEY AUTO_INCREMENT,
    RoleName ENUM('BusinessOwner', 'Admin',
'WarehouseManager') NOT NULL UNIQUE
);

-- 6. Orders Table
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY AUTO_INCREMENT,
    ProductID INT,
    UserID INT,
```

```
    Quantity INT NOT NULL CHECK (Quantity > 0),
    OrderDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Status ENUM('Pending', 'Shipped', 'Delivered', 'Cancelled') DEFAULT 'Pending',
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID) ON DELETE CASCADE,
    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON DELETE SET NULL
);

-- 7. Customers Table
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    PhoneNumber VARCHAR(20),
    Address TEXT
);
```

```
-- 8. Sales Table
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    ProductID INT,
    SaleDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    QuantitySold INT NOT NULL CHECK (QuantitySold > 0),
    TotalPrice DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (CustomerID) REFERENCES
Customers(CustomerID) ON DELETE CASCADE,
    FOREIGN KEY (ProductID) REFERENCES
Products(ProductID) ON DELETE CASCADE
);

-- 9. Inventory Audit Table (Tracks stock changes)
CREATE TABLE Inventory_Audit (
    AuditID INT PRIMARY KEY AUTO_INCREMENT,
    ProductID INT,
```

```
    ActionType ENUM('Stock Added', 'Stock Reduced',
'Damaged') NOT NULL,

    QuantityChanged INT NOT NULL,

    ChangeDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    UserID INT,

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

    FOREIGN KEY (UserID) REFERENCES Users(UserID) ON
DELETE SET NULL

);


-- 10. Warehouses Table

CREATE TABLE Warehouses (

    WarehouseID INT PRIMARY KEY AUTO_INCREMENT,

    Location VARCHAR(255) NOT NULL,

    Capacity INT NOT NULL CHECK (Capacity > 0)

);


-- Warehouse_Stock (Many-to-Many relation between
Warehouses and Products)
```

```
CREATE TABLE Warehouse_Stock (
    WarehouseID INT,
    ProductID INT,
    StockQuantity INT NOT NULL CHECK (StockQuantity >= 0),
    PRIMARY KEY (WarehouseID, ProductID),
    FOREIGN KEY (WarehouseID) REFERENCES
Warehouses(WarehouseID) ON DELETE CASCADE,
    FOREIGN KEY (ProductID) REFERENCES
Products(ProductID) ON DELETE CASCADE
);
```

4)all queries related to inventory/stock management:-

1. Data Insertion Queries (Add Sample Data)

sql

CopyEdit

```
-- 1. Insert more categories
INSERT INTO Categories (CategoryName) VALUES
('Automobile'), ('Grocery'), ('Books');
```

```
-- 2. Insert more suppliers

INSERT INTO Suppliers (Name, ContactInfo, Address)

VALUES ('Supplier C', 'supplierC@example.com', 'Houston'),
('Supplier D', 'supplierD@example.com', 'Seattle');


-- 3. Insert more products

INSERT INTO Products (Name, CategoryID, StockLevel, Price,
ReorderLevel, SupplierID)

VALUES

('Car Battery', 4, 20, 150.00, 5, 3),

('Milk', 5, 100, 2.00, 10, 4),

('Notebook', 6, 500, 5.00, 30, 3);


-- 4. Insert more customers

INSERT INTO Customers (Name, Email, PhoneNumber,
Address)

VALUES

('Emma Watson', 'emma@example.com', '1122334455',
'London'),
```

('Robert Downey', 'robert@example.com', '2233445566', 'New York');

-- 5. Insert more orders

INSERT INTO Orders (ProductID, UserID, Quantity, Status)

VALUES (4, 1, 5, 'Pending'), (5, 2, 2, 'Shipped');

-- 6. Insert more sales

INSERT INTO Sales (CustomerID, ProductID, QuantitySold, TotalPrice)

VALUES

(3, 4, 2, 300.00),

(4, 5, 3, 6.00);

---

2. Data Retrieval Queries (Basic SELECT Queries)

sql

CopyEdit

-- 7. Get all orders placed by a specific user

```
SELECT * FROM Orders WHERE UserID = 1;


-- 8. List all orders along with product details

SELECT Orders.OrderID, Products.Name, Orders.Quantity,
Orders.Status

FROM Orders

INNER JOIN Products ON Orders.ProductID =
Products.ProductID;


-- 9. Get all products along with their supplier names

SELECT Products.Name, Suppliers.Name AS Supplier

FROM Products

INNER JOIN Suppliers ON Products.SupplierID =
Suppliers.SupplierID;


-- 10. List all orders along with user details

SELECT Orders.OrderID, Users.Username, Orders.Quantity,
Orders.Status

FROM Orders

INNER JOIN Users ON Orders.UserID = Users.UserID;
```

---

3. Analytical Queries (SUM, COUNT, AVG, MAX, MIN, GROUP BY)

sql

CopyEdit

-- 11. Get total number of products

SELECT COUNT(*) AS TotalProducts FROM Products;

-- 12. Get total revenue generated

SELECT SUM(TotalPrice) AS TotalRevenue FROM Sales;

-- 13. Get average product price

SELECT AVG(Price) AS AvgPrice FROM Products;

-- 14. Get the most expensive product

SELECT Name, Price FROM Products ORDER BY Price DESC LIMIT 1;

-- 15. Get total sales for each product

SELECT ProductID, SUM(QuantitySold) AS TotalSold FROM Sales GROUP BY ProductID;

---

4. JOIN Queries (Combining Data Across Tables)

sql

CopyEdit

-- 16. Get sales details including customer name and product name

SELECT Sales.SaleID, Customers.Name AS Customer, Products.Name AS Product, Sales.QuantitySold, Sales.TotalPrice

FROM Sales

INNER JOIN Customers ON Sales.CustomerID = Customers.CustomerID

INNER JOIN Products ON Sales.ProductID = Products.ProductID;

-- 17. Get all warehouse stock details

```
SELECT Warehouses.Location, Products.Name,
Warehouse_Stock.StockQuantity

FROM Warehouse_Stock

INNER JOIN Warehouses ON Warehouse_Stock.WarehouseID
= Warehouses.WarehouseID

INNER JOIN Products ON Warehouse_Stock.ProductID =
Products.ProductID;


-- 18. List suppliers and the products they provide

SELECT Suppliers.Name AS Supplier, Products.Name AS
Product

FROM Suppliers

INNER JOIN Products ON Suppliers.SupplierID =
Products.SupplierID;
```

---

5. Advanced Queries (Subqueries, Window Functions, and Views)

sql

CopyEdit

```
-- 19. Get the product with the highest sales

SELECT ProductID, SUM(QuantitySold) AS TotalSales

FROM Sales

GROUP BY ProductID

ORDER BY TotalSales DESC

LIMIT 1;


-- 20. Get the number of products supplied by each supplier

SELECT SupplierID, COUNT(*) AS ProductCount FROM
Products GROUP BY SupplierID;


-- 21. View for active orders

CREATE VIEW ActiveOrders AS

SELECT OrderID, ProductID, Quantity, Status

FROM Orders WHERE Status IN ('Pending', 'Shipped');


-- 22. Get stock status of each product in warehouses

SELECT Products.Name,
SUM(Warehouse_Stock.StockQuantity) AS TotalStock
```

```
FROM Warehouse_Stock

INNER JOIN Products ON Warehouse_Stock.ProductID =
Products.ProductID

GROUP BY Products.Name;
```

---

6. Updating Data (Modifying Existing Entries)

sql

CopyEdit

```
-- 23. Update stock level after a sale

UPDATE Products SET StockLevel = StockLevel - 5 WHERE
ProductID = 2;


-- 24. Update an order status

UPDATE Orders SET Status = 'Delivered' WHERE OrderID = 1;
```

---

7. Deleting Data (Removing Entries Safely)

sql

CopyEdit

```
-- 25. Delete an inactive customer
DELETE FROM Customers WHERE CustomerID = 6;


-- 26. Remove an order that was canceled
DELETE FROM Orders WHERE Status = 'Cancelled';
```

---

8. Stored Procedures and Functions

sql

CopyEdit

```
-- 27. Stored Procedure to check low stock
DELIMITER //
CREATE PROCEDURE CheckLowStock()
BEGIN
    SELECT * FROM Products WHERE StockLevel <=
ReorderLevel;
END //
```

```
DELIMITER ;

-- Call Procedure
CALL CheckLowStock();
```

---

9. Transactions (Ensuring Data Integrity)

sql

CopyEdit

```
-- 28. Create a transaction to process an order
START TRANSACTION;

UPDATE Products SET StockLevel = StockLevel - 3 WHERE ProductID = 2;
INSERT INTO Orders (ProductID, UserID, Quantity, Status) VALUES (2, 1, 3, 'Pending');

COMMIT;
```

---

10. Triggers (Automating Processes)

sql

CopyEdit

```
-- 29. Create a trigger to log stock updates

DELIMITER //

CREATE TRIGGER after_stock_update

AFTER UPDATE ON Products

FOR EACH ROW

BEGIN

    INSERT INTO Inventory_Audit (ProductID, ActionType, QuantityChanged, ChangeDate, UserID)

    VALUES (NEW.ProductID, 'Stock Updated', (NEW.StockLevel - OLD.StockLevel), NOW(), 1);

END //

DELIMITER ;
```

1. Advanced Data Retrieval Queries (Filtering, Sorting, Limits)

sql

CopyEdit

-- 101. Get the latest 10 orders

SELECT * FROM Orders ORDER BY OrderDate DESC LIMIT 10;

-- 102. Get all orders placed within the last 7 days

SELECT * FROM Orders WHERE OrderDate >= CURDATE() - INTERVAL 7 DAY;

-- 103. Get products with stock level between 10 and 50

SELECT * FROM Products WHERE StockLevel BETWEEN 10 AND 50;

-- 104. Get products costing more than $100

SELECT * FROM Products WHERE Price > 100;

-- 105. Get all products from a specific category (e.g., Electronics)

SELECT * FROM Products WHERE CategoryID = (SELECT CategoryID FROM Categories WHERE CategoryName = 'Electronics');

---

2. Financial Analysis Queries (Revenue, Profit, Discounts, Taxes)

sql

CopyEdit

```
-- 106. Calculate total revenue per product

SELECT ProductID, SUM(TotalPrice) AS Revenue FROM Sales GROUP BY ProductID;


-- 107. Calculate total revenue for each customer

SELECT CustomerID, SUM(TotalPrice) AS TotalSpent FROM Sales GROUP BY CustomerID;


-- 108. Calculate monthly sales revenue

SELECT MONTH(OrderDate) AS Month, SUM(TotalPrice) AS Revenue FROM Sales GROUP BY MONTH(OrderDate);


-- 109. Get the most profitable products (highest revenue)
```

```
SELECT ProductID, SUM(TotalPrice) AS TotalRevenue FROM Sales GROUP BY ProductID ORDER BY TotalRevenue DESC LIMIT 5;


-- 110. Calculate tax collected on sales (assuming 10% tax rate)

SELECT SUM(TotalPrice * 0.10) AS TotalTax FROM Sales;
```

---

3. Inventory Tracking Queries (Stock Alerts, Movement Analysis)

sql

CopyEdit

```
-- 111. Get all products that need to be restocked

SELECT * FROM Products WHERE StockLevel <= ReorderLevel;


-- 112. Get stock levels for all products in a specific warehouse

SELECT Products.Name, Warehouse_Stock.StockQuantity
```

```
FROM Warehouse_Stock

INNER JOIN Products ON Warehouse_Stock.ProductID =
Products.ProductID

WHERE WarehouseID = 1;


-- 113. Get stock movement history for a product

SELECT * FROM Inventory_Audit WHERE ProductID = 2
ORDER BY ChangeDate DESC;


-- 114. Get the most frequently restocked product

SELECT ProductID, COUNT(*) AS RestockCount FROM
Inventory_Audit WHERE ActionType = 'Stock Added' GROUP
BY ProductID ORDER BY RestockCount DESC LIMIT 1;


-- 115. Find products that have not been sold in the last 3
months

SELECT * FROM Products WHERE ProductID NOT IN (SELECT
DISTINCT ProductID FROM Sales WHERE SaleDate >=
CURDATE() - INTERVAL 3 MONTH);
```

---

4. Customer & Order Analytics (Behavior, Trends, Loyalty)

sql

CopyEdit

-- 116. Get the top 5 customers who made the highest purchases

SELECT CustomerID, SUM(TotalPrice) AS TotalSpent FROM Sales GROUP BY CustomerID ORDER BY TotalSpent DESC LIMIT 5;

-- 117. Find repeat customers (customers who have placed more than 3 orders)

SELECT CustomerID, COUNT(*) AS OrderCount FROM Orders GROUP BY CustomerID HAVING OrderCount > 3;

-- 118. Get customers who have not made a purchase in the last 6 months

SELECT * FROM Customers WHERE CustomerID NOT IN (SELECT DISTINCT CustomerID FROM Sales WHERE SaleDate >= CURDATE() - INTERVAL 6 MONTH);

```
-- 119. Get products that are frequently bought together (based on past orders)

SELECT o1.ProductID AS Product_A, o2.ProductID AS Product_B, COUNT(*) AS Frequency

FROM Orders o1

JOIN Orders o2 ON o1.OrderID = o2.OrderID AND o1.ProductID < o2.ProductID

GROUP BY o1.ProductID, o2.ProductID

ORDER BY Frequency DESC

LIMIT 5;


-- 120. Get the average order value per customer

SELECT CustomerID, AVG(TotalPrice) AS AvgOrderValue FROM Sales GROUP BY CustomerID;
```

---

5. Performance Optimization Queries (Indexes, Caching, Query Tuning)

sql

CopyEdit

-- 121. Create an index on the Orders table to speed up searches

CREATE INDEX idx_orders_date ON Orders (OrderDate);

-- 122. Optimize product searches by indexing the product name

CREATE INDEX idx_products_name ON Products (Name);

-- 123. Find slow queries in MySQL performance schema

SELECT * FROM performance_schema.events_statements_summary_by_digest ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;

-- 124. Get the most accessed products (based on sales frequency)

SELECT ProductID, COUNT(*) AS PurchaseCount FROM Sales GROUP BY ProductID ORDER BY PurchaseCount DESC LIMIT 10;

-- 125. Create a materialized view for frequent reports (if using MySQL 8+)

```
CREATE TABLE Sales_Summary AS

SELECT ProductID, SUM(TotalPrice) AS Revenue,
SUM(QuantitySold) AS TotalSold FROM Sales GROUP BY
ProductID;
```

---

6. Security Queries (Access Control, Audit Logs, Role-Based Permissions)

sql

CopyEdit

```
-- 126. Create a new role for Inventory Managers

CREATE ROLE InventoryManager;


-- 127. Grant permissions to the Inventory Manager role

GRANT SELECT, UPDATE, INSERT ON Products TO
InventoryManager;


-- 128. Assign the Inventory Manager role to a user

GRANT InventoryManager TO 'user_inventory';
```

-- 129. Track login attempts and failures

SELECT * FROM mysql.general_log WHERE argument LIKE '%login%' ORDER BY event_time DESC LIMIT 5;

-- 130. Log actions of admins (audit trail)

SELECT * FROM Inventory_Audit WHERE UserID IN (SELECT UserID FROM Users WHERE RoleID = (SELECT RoleID FROM Roles WHERE RoleName = 'Admin'));

---

7. Warehouse & Supplier Management Queries

sql

CopyEdit

-- 131. Get the total stock stored in each warehouse

SELECT WarehouseID, SUM(StockQuantity) AS TotalStock FROM Warehouse_Stock GROUP BY WarehouseID;

-- 132. Get a list of suppliers and the total number of products they provide

```
SELECT SupplierID, COUNT(ProductID) AS ProductCount
FROM Products GROUP BY SupplierID;


-- 133. Get the supplier with the most product shipments

SELECT SupplierID, COUNT(*) AS Shipments FROM Orders
GROUP BY SupplierID ORDER BY Shipments DESC LIMIT 1;


-- 134. Get warehouse capacity utilization percentage

SELECT WarehouseID, SUM(StockQuantity) / Capacity * 100
AS UtilizationPercentage FROM Warehouse_Stock INNER
JOIN Warehouses ON Warehouse_Stock.WarehouseID =
Warehouses.WarehouseID GROUP BY WarehouseID;


-- 135. Identify under-utilized warehouses (less than 50% full)

SELECT WarehouseID FROM (SELECT WarehouseID,
SUM(StockQuantity) / Capacity * 100 AS Utilization FROM
Warehouse_Stock INNER JOIN Warehouses ON
Warehouse_Stock.WarehouseID = Warehouses.WarehouseID
GROUP BY WarehouseID) AS UtilizationTable WHERE
Utilization < 50;
```