**✅ Inventory and Warehouse Management System – Project Documentation**

**🎯 Objective**

To design and implement a real-world backend SQL system for managing warehouse inventory, tracking product stock across multiple locations, sending low-stock alerts, and automating inter-warehouse stock transfers.

**🛠️ Tools & Technologies Used**

| **Tool** | **Purpose** |
| --- | --- |
| MySQL | Relational Database Management System |
| DBeaver | SQL development and visualization tool (GUI) |
| SQL | Query Language for defining and manipulating data |

**🗃️ What You Built**

You designed and developed a structured SQL-based Inventory Management System consisting of:

**📦 Core Tables**

1. **Supplier**
   * Stores supplier details
   * Fields: supplier\_id, supplier\_name, contact\_email
2. **Product**
   * Stores product information
   * Linked to suppliers
   * Fields: product\_id, product\_name, unit\_price, reorder\_level, supplier\_id
3. **Warehouse**
   * Stores warehouse locations
   * Fields: warehouse\_id, warehouse\_name, location
4. **Stock**
   * Tracks product quantity per warehouse (many-to-many relation)
   * Fields: stock\_id, product\_id, warehouse\_id, quantity

**🔗 Relationships**

| **Relationship** | **Description** |
| --- | --- |
| Supplier → Product | One-to-Many |
| Product ↔ Warehouse | Many-to-Many (via Stock table) |

**🧪 Sample Data Inserted**

* **2 Suppliers**: TechSource, MegaParts
* **3 Products**: Laptop, Mouse, Keyboard
* **2 Warehouses**: Central, Regional
* **5 Stock Entries**: Linking products to warehouse inventories

**🔍 Views Created**

**📄 inventory\_status**

Shows current product quantity in each warehouse.

CREATE VIEW inventory\_status AS

SELECT p.product\_name, w.warehouse\_name, s.quantity, p.reorder\_level

FROM Stock s

JOIN Product p ON s.product\_id = p.product\_id

JOIN Warehouse w ON s.warehouse\_id = w.warehouse\_id;

**🔔 Trigger: Low Stock Alert**

You implemented an AFTER INSERT trigger that:

* Checks if inserted stock is below product's reorder level
* Raises an error message using SIGNAL SQLSTATE

CREATE TRIGGER low\_stock\_alert

AFTER INSERT ON Stock

FOR EACH ROW

BEGIN

DECLARE reorder\_threshold INT;

SELECT reorder\_level INTO reorder\_threshold FROM Product WHERE product\_id = NEW.product\_id;

IF NEW.quantity < reorder\_threshold THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = CONCAT('⚠️ LOW STOCK ALERT: ', NEW.quantity, ' units is below reorder level for Product ID ', NEW.product\_id);

END IF;

END;

**🔁 Stored Procedure: Stock Transfer**

You wrote a stored procedure to:

* Decrease stock from one warehouse
* Increase the same stock at another warehouse
* Create a new record if the destination doesn’t already have it

CREATE PROCEDURE transfer\_stock (

IN prod\_id INT,

IN from\_wh INT,

IN to\_wh INT,

IN qty INT

)

BEGIN

-- Subtract from source

UPDATE Stock

SET quantity = quantity - qty

WHERE product\_id = prod\_id AND warehouse\_id = from\_wh;

-- Add to destination

INSERT INTO Stock (product\_id, warehouse\_id, quantity)

VALUES (prod\_id, to\_wh, qty)

ON DUPLICATE KEY UPDATE quantity = quantity + qty;

END;

**📊 Useful Reports & Queries**

**1. Show current inventory**

SELECT \* FROM inventory\_status;

**2. Products below reorder level**

SELECT p.product\_name, s.quantity

FROM Stock s

JOIN Product p ON s.product\_id = p.product\_id

WHERE s.quantity < p.reorder\_level;

**3. Total quantity per product (all warehouses)**

SELECT p.product\_name, SUM(s.quantity) AS total\_quantity

FROM Stock s

JOIN Product p ON s.product\_id = p.product\_id

GROUP BY p.product\_name;

**4. Perform stock transfer (example)**

CALL transfer\_stock(1, 1, 2, 2); -- Transfer 2 laptops from warehouse 1 to 2

**🧠 Concepts Practiced**

| **Concept** | **Description** |
| --- | --- |
| ER Design | Normalized schema and many-to-many modeling |
| Triggers | Automation using SQL logic |
| Stored Procedures | Reusable logic for stock operations |
| Views | Simplified data presentation |
| Joins & Queries | Reporting and inventory status retrieval |