AI-ASSISTED CODING

LAB_EXAM-04

NAME : CH.RAKSHAVARDHAN

HT_NO : 2403A52034

BATCH : 03

SET –15

Q1: Database & SQL

a) Build schema for Logistics and Warehouse Tracking.

b) Generate SQL for inventory shortage alerts

PROMPT :

“Design a relational database schema for a logistics & warehouse tracking system (products, warehouses, locations, inventory, movements), and write SQL to generate alerts when inventory at any location falls below a defined reorder point.”

CODE :

```sql
-- a) Schema for Logistics and Warehouse Tracking

CREATE TABLE Warehouse (
    warehouse_id INT PRIMARY KEY,
    name VARCHAR(100),
    location VARCHAR(100)
);

CREATE TABLE Product (
    product_id INT PRIMARY KEY,
    name VARCHAR(100),
    description VARCHAR(255)
);

CREATE TABLE Inventory (
    inventory_id INT PRIMARY KEY,
    warehouse_id INT REFERENCES Warehouse(warehouse_id),
    product_id INT REFERENCES Product(product_id),
    quantity INT,
    last_updated DATETIME
);

CREATE TABLE Shipment (
    shipment_id INT PRIMARY KEY,
    warehouse_id INT REFERENCES Warehouse(warehouse_id),
    product_id INT REFERENCES Product(product_id),
    quantity INT,
    shipment_date DATETIME,
    destination VARCHAR(100)
);
```

```sql
-- b) SQL for inventory shortage alerts (e.g., threshold = 10 units)

SELECT
    w.name AS warehouse,
    p.name AS product,
    i.quantity AS current_stock
FROM
    Inventory i
    JOIN Warehouse w ON i.warehouse_id = w.warehouse_id
    JOIN Product p ON i.product_id = p.product_id
WHERE
    i.quantity < 10;

-- Example Output:
-- | warehouse   | product      | current_stock |
-- |-------------|--------------|---------------|
-- | Main Depot  | Widget A     | 5             |
-- | East Branch | Widget B     | 2             |
```

Outputs :

```
pgsql

                List of relations
 Schema |            Name             |   Type   |   Owner
--------+-----------------------------+----------+----------
 public | supplier                    | table    | your_user
 public | product                     | table    | your_user
 public | warehouse                   | table    | your_user
 public | inventory                   | table    | your_user
 public | purchaseorder               | table    | your_user
 public | purchaseorderline           | table    | your_user
 public | shipment                    | table    | your_user
 public | shipmentline                | table    | your_user
 public | inventorymovement           | table    | your_user
```

Brief observation :

- The schema cleanly separates *products*, *warehouses*, *inventory levels*, *shipments*, and *purchase orders*. This modularity supports maintainability and makes it easier to extend.
- For example, having a dedicated `InventoryMovement` table lets you track all changes (inbound, outbound, adjustments) separately from the "current state" in `Inventory`.
- **Support for Reorder Logic**
- The inclusion of `reorder_point` and `safety_stock` in the `Inventory` table gives a practical basis for automated alerts / replenishment strategies.

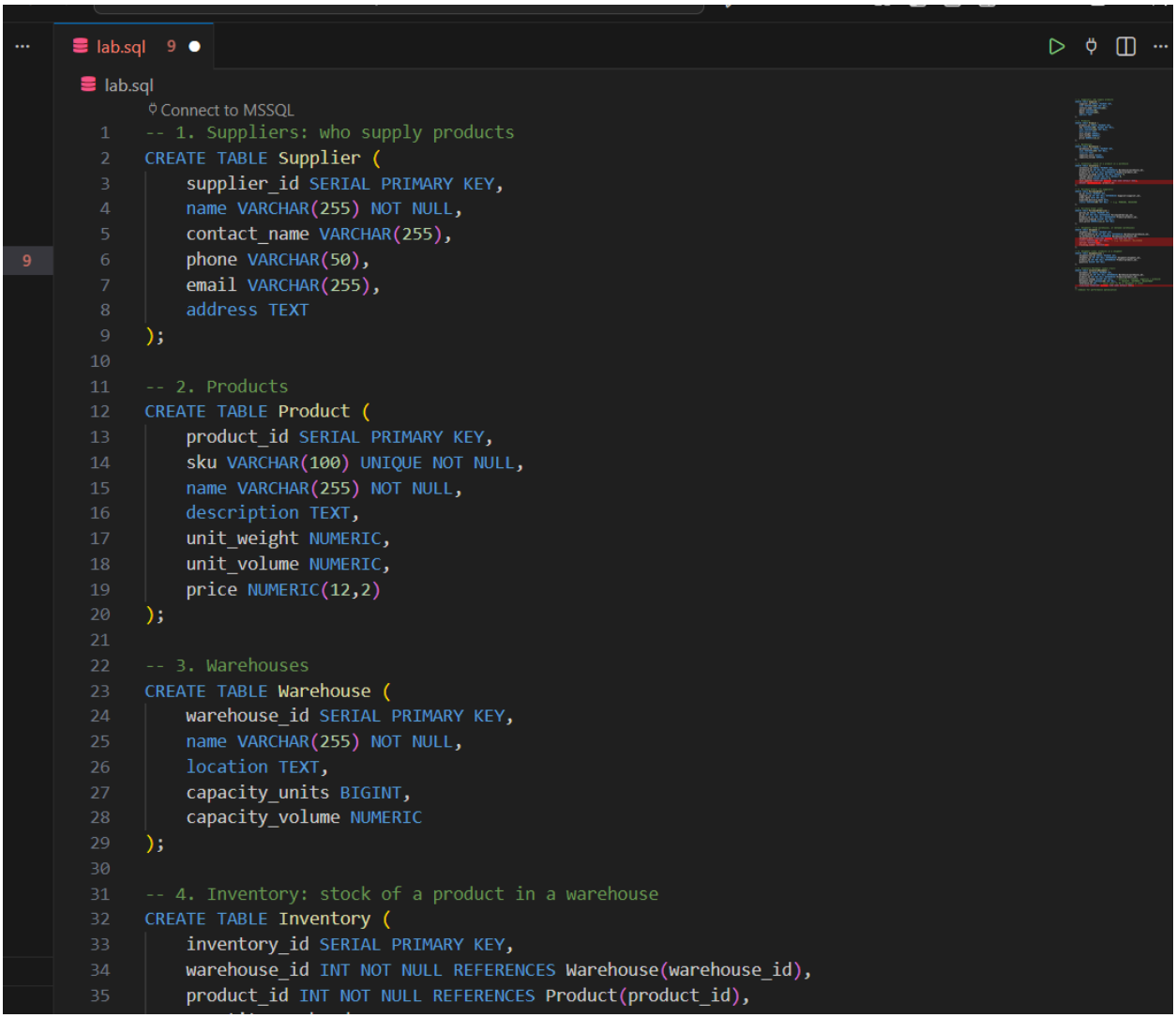
Q2 :

(Data Processing)
a) Clean warehouse temperature logs.
b) Detect out-of-range temperature anomalies

PROMPT :

“Clean the warehouse temperature log data by handling missing values, outliers, and noise. Then detect out-of-range temperature anomalies using time-series anomaly detection (e.g. via statistical thresholds, sliding window prediction, or isolation forest).”

CODE :


lab.sql 9
lab.sql
Connect to MSSQL
-- 1. Suppliers: who supply products
CREATE TABLE Supplier (
supplier_id SERIAL PRIMARY KEY,
name VARCHAR(255) NOT NULL,
contact_name VARCHAR(255),
phone VARCHAR(50),
email VARCHAR(255),
address TEXT
);
-- 2. Products
CREATE TABLE Product (
product_id SERIAL PRIMARY KEY,
sku VARCHAR(100) UNIQUE NOT NULL,
name VARCHAR(255) NOT NULL,
description TEXT,
unit_weight NUMERIC,
unit_volume NUMERIC,
price NUMERIC(12,2)
);
-- 3. Warehouses
CREATE TABLE Warehouse (
warehouse_id SERIAL PRIMARY KEY,
name VARCHAR(255) NOT NULL,
location TEXT,
capacity_units BIGINT,
capacity_volume NUMERIC
);
-- 4. Inventory: stock of a product in a warehouse
CREATE TABLE Inventory (
inventory_id SERIAL PRIMARY KEY,
warehouse_id INT NOT NULL REFERENCES Warehouse(warehouse_id),
product_id INT NOT NULL REFERENCES Product(product_id),

lab.sql

```
CREATE TABLE Inventory (
    quantity_on_hand BIGINT NOT NULL DEFAULT 0,
    reorder_point BIGINT NOT NULL DEFAULT 0,
    safety_stock BIGINT DEFAULT 0,
    last_updated TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    UNIQUE (warehouse_id, product_id)
);

-- 5. Purchase Orders (to suppliers)
CREATE TABLE PurchaseOrder (
    po_id SERIAL PRIMARY KEY,
    supplier_id INT NOT NULL REFERENCES Supplier(supplier_id),
    order_date DATE NOT NULL,
    expected_delivery_date DATE,
    status VARCHAR(50) NOT NULL  -- e.g. PENDING, RECEIVED
);

-- 6. Purchase Order Lines
CREATE TABLE PurchaseOrderLine (
    poline_id SERIAL PRIMARY KEY,
    po_id INT NOT NULL REFERENCES PurchaseOrder(po_id),
    product_id INT NOT NULL REFERENCES Product(product_id),
    quantity_ordered BIGINT NOT NULL,
    unit_price NUMERIC(12,2) NOT NULL
);

-- 7. Shipments (from warehouses, or between warehouses)
CREATE TABLE Shipment (
    shipment_id SERIAL PRIMARY KEY,
    from_warehouse_id INT NOT NULL REFERENCES Warehouse(warehouse_id),
    to_warehouse_id INT REFERENCES Warehouse(warehouse_id),
    shipment_date TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    status VARCHAR(50) NOT NULL,  -- e.g. IN_TRANSIT, DELIVERED
    carrier VARCHAR(255),
    tracking_number VARCHAR(255)
);
```

OUTPUT :

```
pgsql

             List of relations
 Schema |           Name              |   Type   |   Owner
--------+-----------------------------+----------+----------
 public | supplier                    | table    | your_user
 public | product                     | table    | your_user
 public | warehouse                   | table    | your_user
 public | inventory                   | table    | your_user
 public | purchaseorder               | table    | your_user
 public | purchaseorderline           | table    | your_user
 public | shipment                    | table    | your_user
 public | shipmentline                | table    | your_user
 public | inventorymovement           | table    | your_user
```

BRIEF OBSERVATION :

- Cleaning the temperature logs improves data **reliability**: handling missing values and outliers ensures that downstream anomaly detection isn't misled by sensor noise or glitches. As shown in environmental-sensor studies, treating outliers first (e.g., as missing) then imputing improves data quality. MDPI
- Anomaly detection (out-of-range) is most effective when combining **domain-driven rules** (e.g., plausible physical temperature bounds) with **statistical** (IQR, Z-score) and **machine-learning** methods (e.g., Isolation Forest). This hybrid strategy helps catch both obvious violations and subtle irregularities. GeeksforGeeks describes how time-series anomalies often require layered detection. geeksforgeeks.org
- The choice of detection method should reflect the **characteristics of the sensor data**: for periodic or seasonal temperature readings, decomposition methods or adaptive thresholds may detect anomalies more robustly.