DISTRIBUTED DATABASE SYSTEMS

PROJECT
“INTELLIGENT SUPPLY CHAIN DECISION SYSTEM”

Members:

Tahrim Bilal (B21110006153)
Rimsha Laraib (B21110006107)
Mehak Fatima (B21110006057)
Yumna Mubeen (B21110006165)

Submitted to:

Sir Zaeem Tariq

Department:

Department of Computer Science (UBIT)

Intelligent Supply Chain Decision System

Project Description:

Design and implement a **database-driven decision support system** that manages and optimizes a **multi-level supply chain network**.
The system must model **suppliers, warehouses, retailers, shipments, and orders**, while dynamically determining **optimal paths**, **stock allocations**, and **restocking strategies** — purely using **SQL logic and database design**, not application code.
This project requires students to use **recursive queries**, **aggregate analysis**, **trigger-based reasoning**, and **constraint-based logic** to simulate real-world decision-making.

Scenario:

A company manages a three-tier supply network:

- **Suppliers** deliver raw materials to **warehouses**.
- **Warehouses** distribute products to **retailers**.
- **Retailers** serve **customers** and place restocking orders.

Each order travels through multiple hops. Students must build a database that can:

1. Database and table creation:

```sql
CREATE DATABASE supply_chain_project;
USE supply_chain_project;


-- Supplier Table
CREATE TABLE Supplier (
    supplier_id INT AUTO_INCREMENT PRIMARY KEY,
    supplier_name VARCHAR(100),
    location VARCHAR(100)
);
-- Warehouse Table
CREATE TABLE Warehouse (
    warehouse_id INT AUTO_INCREMENT PRIMARY KEY,
    warehouse_name VARCHAR(100),
    location VARCHAR(100)
);
-- Retailer Table
CREATE TABLE Retailer (
    retailer_id INT AUTO_INCREMENT PRIMARY KEY,
    retailer_name VARCHAR(100),
    location VARCHAR(100)
);
```

```sql
-- Product Table
CREATE TABLE Product (
    product_id INT AUTO_INCREMENT PRIMARY KEY,
    product_name VARCHAR(100)
);
-- Inventory Table (for supplier/warehouse/retailer)
CREATE TABLE Inventory (
    inventory_id INT AUTO_INCREMENT PRIMARY KEY,
    node_type ENUM('SUPPLIER','WAREHOUSE','RETAILER'),
    node_id INT,
    product_id INT,
    quantity INT DEFAULT 0,
    threshold INT DEFAULT 20,
    FOREIGN KEY (product_id) REFERENCES Product(product_id)
);
-- Route Table (multi-hop network)
CREATE TABLE Route (
    route_id INT AUTO_INCREMENT PRIMARY KEY,
    from_node_type ENUM('SUPPLIER','WAREHOUSE','RETAILER'),
    from_node_id INT,
    to_node_type ENUM('SUPPLIER','WAREHOUSE','RETAILER'),
    to_node_id INT,
    transport_cost DECIMAL(10,2)
);
-- Orders Table
CREATE TABLE Orders (
    order_id INT AUTO_INCREMENT PRIMARY KEY,
    retailer_id INT,
    product_id INT,
    quantity INT,
    deadline DATE,
    status VARCHAR(20) DEFAULT 'PENDING',
    FOREIGN KEY (retailer_id) REFERENCES Retailer(retailer_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id)
);
-- Shipment Table
CREATE TABLE Shipment (
    shipment_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT,
    from_node_type ENUM('SUPPLIER','WAREHOUSE'),
    from_node_id INT,
    to_node_type ENUM('WAREHOUSE','RETAILER'),
    to_node_id INT,
    product_id INT,
    quantity INT,
    cost DECIMAL(10,2),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
-- Restock Requests
CREATE TABLE Restock_Request (
    request_id INT AUTO_INCREMENT PRIMARY KEY,
    node_type ENUM('SUPPLIER','WAREHOUSE','RETAILER'),
    node_id INT,
    product_id INT,
    current_quantity INT,
    request_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

1. Track inventory levels at every node.

```sql
-- 1. Track inventory levels at every node
INSERT INTO Inventory (node_type, node_id, product_id, quantity, threshold)
VALUES
('SUPPLIER', 1, 1, 500, 50),
('SUPPLIER', 2, 2, 400, 50),
('WAREHOUSE', 1, 1, 100, 20),
('WAREHOUSE', 2, 2, 80, 20),
('RETAILER', 1, 1, 10, 5),
('RETAILER', 2, 2, 5, 5),
('WAREHOUSE', 3, 3, 50, 10),
('RETAILER', 3, 3, 15, 5),
('SUPPLIER', 3, 3, 300, 30),
('WAREHOUSE', 4, 4, 70, 20);
```

1. Record shipments and their paths.

```sql
-- 2. Record shipments and their paths
INSERT INTO Route (from_node_type, from_node_id, to_node_type, to_node_id, transport_cost)
VALUES
('SUPPLIER', 1, 'WAREHOUSE', 1, 50),
('SUPPLIER', 2, 'WAREHOUSE', 2, 60),
('WAREHOUSE', 1, 'RETAILER', 1, 30),
('WAREHOUSE', 2, 'RETAILER', 2, 40),
('SUPPLIER', 3, 'WAREHOUSE', 3, 45),
('WAREHOUSE', 3, 'RETAILER', 3, 35),
('SUPPLIER', 4, 'WAREHOUSE', 4, 55),
('WAREHOUSE', 4, 'RETAILER', 4, 25),
('SUPPLIER', 5, 'WAREHOUSE', 5, 65),
('WAREHOUSE', 5, 'RETAILER', 5, 30);
INSERT INTO Shipment (order_id, from_node_type, from_node_id, to_node_type, to_node_id, product_id,
quantity, cost)
VALUES
(1, 'SUPPLIER', 1, 'WAREHOUSE', 1, 1, 5, 50),
(2, 'SUPPLIER', 2, 'WAREHOUSE', 2, 2, 10, 60),
(3, 'SUPPLIER', 3, 'WAREHOUSE', 3, 3, 7, 45),
(4, 'SUPPLIER', 4, 'WAREHOUSE', 4, 4, 12, 55),
(5, 'SUPPLIER', 5, 'WAREHOUSE', 5, 5, 20, 65);
```

2. Automatically detect supply shortages and generate restock suggestions.

```sql
-- 3. Automatically detect supply shortages and generate restock suggestions
DELIMITER $$

CREATE TRIGGER trg_low_stock
AFTER UPDATE ON Inventory
FOR EACH ROW
BEGIN
    IF NEW.quantity < NEW.threshold THEN
        INSERT INTO Restock_Request (node_type, node_id, product_id, current_quantity)
        VALUES (NEW.node_type, NEW.node_id, NEW.product_id, NEW.quantity);
    END IF;
END $$
DELIMITER ;
-- Test Storage
UPDATE Inventory
SET quantity = 2
WHERE node_type='RETAILER' AND node_id=1 AND product_id=1;
-- Check
SELECT * FROM Restock_Request;
```

request_id	node_type	node_id	product_id	current_quantity	request_date
1	WAREHOUSE	1	1	100	2025-11-29 17:48:25
2	WAREHOUSE	2	2	80	2025-11-29 17:48:25
3	RETAILER	1	1	10	2025-11-29 17:48:25
4	RETAILER	2	2	5	2025-11-29 17:48:25
5	WAREHOUSE	3	3	50	2025-11-29 17:48:25
6	RETAILER	1	1	2	2025-11-29 18:00:30

4. Compute the **shortest-cost route** between supplier and retailer (using recursive SQL or advanced join logic).

```sql
-- 4. Compute the shortest-cost route between supplier and retailer
-- This query finds the cheapest path and total transport cost. for best path
WITH RECURSIVE SupplyRoutes AS (
    SELECT route_id, from_node_type, from_node_id, to_node_type, to_node_id, transport_cost,
    transport_cost AS total_cost
    FROM Route
    WHERE from_node_type='SUPPLIER' AND from_node_id=1
    UNION ALL
    SELECT r.route_id, r.from_node_type, r.from_node_id, r.to_node_type, r.to_node_id, r.transport_cost,
    sr.total_cost + r.transport_cost
    FROM Route r
    INNER JOIN SupplyRoutes sr
        ON r.from_node_type = sr.to_node_type AND r.from_node_id = sr.to_node_id
)
SELECT *
FROM SupplyRoutes
WHERE to_node_type='RETAILER' AND to_node_id=1
ORDER BY total_cost
LIMIT 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

route_id	from_node_type	from_node_id	to_node_type	to_node_id	transport_cost	total_cost
3	WAREHOUSE	1	RETAILER	1	30.00	80.00

5. Prioritize urgent orders based on deadlines and available stock.

```sql
-- 5. Prioritize urgent orders based on deadlines and available stock
-- This will show urgent orders first and whether you have enough stock.
SELECT o.order_id, o.retailer_id, o.product_id, o.quantity, o.deadline,
       i.quantity AS available_stock,
       CASE WHEN i.quantity >= o.quantity THEN 'Can Fulfill' ELSE 'Stock Short' END AS status
FROM Orders o
JOIN Inventory i
ON i.node_type='WAREHOUSE' AND i.product_id = o.product_id
ORDER BY o.deadline ASC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	retailer_id	product_id	quantity	deadline	available_stock	status
1	1	1	5	2025-12-05	-2	Stock Short
2	2	2	10	2025-12-06	80	Can Fulfill
3	3	3	7	2025-12-07	50	Can Fulfill
4	4	4	12	2025-12-08	70	Can Fulfill

2. Complex Logic Requirements

Students must **express reasoning in SQL**, not in procedural code.
Examples:

● Detect all retailers that cannot fulfill orders due to low inventory.

```
-- Detect all retailers that cannot fulfill orders due to low inventory
SELECT o.order_id,
       r.retailer_name,o.product_id,p.product_name,
       o.quantity AS order_quantity,
       SUM(i.quantity) AS total_warehouse_stock,
       CASE
           WHEN SUM(i.quantity) < o.quantity THEN 'Cannot Fulfill'
           ELSE 'Can Fulfill'
       END AS fulfillment_status
FROM Orders o
JOIN Retailer r ON o.retailer_id = r.retailer_id
JOIN Product p ON o.product_id = p.product_id
JOIN Inventory i ON i.node_type='WAREHOUSE' AND i.product_id = o.product_id
GROUP BY o.order_id
HAVING total_warehouse_stock < o.quantity;
-- Test
-- Suppose Warehouse 1 has only 3 units of product 1 instead of 100.
-- Update inventory to simulate low stock:
UPDATE Inventory
SET quantity = 3
WHERE node_type='WAREHOUSE' AND node_id = 1 AND product_id = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	retailer_name	product_id	product_name	order_quantity	total_warehouse_stock	fulfillment_status
1	Retailer R1	1	Steel Rod	5	-2	Cannot Fulfill

● Find **alternative fulfillment paths** (e.g., another warehouse can serve that retailer) using recursive CTEs.

```
-- for alternative path
WITH RECURSIVE SupplyRoutes AS (
    SELECT route_id, from_node_type, from_node_id, to_node_type, to_node_id, transport_cost,
    transport_cost AS total_cost,
           CONCAT(from_node_type,'-',from_node_id,'->',to_node_type,'-',to_node_id) AS path
    FROM Route
    WHERE from_node_type='SUPPLIER' AND from_node_id=1

    UNION ALL

    SELECT r.route_id, r.from_node_type, r.from_node_id, r.to_node_type, r.to_node_id, r.transport_cost,
    sr.total_cost + r.transport_cost,
           CONCAT(sr.path,'->',r.to_node_type,'-',r.to_node_id) AS path
    FROM Route r
    INNER JOIN SupplyRoutes sr
        ON r.from_node_type = sr.to_node_type AND r.from_node_id = sr.to_node_id
)
SELECT *
FROM SupplyRoutes
WHERE to_node_type='RETAILER' AND to_node_id=1
ORDER BY total_cost;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id	from_node_type	from_node_id	to_node_type	to_node_id	transport_cost	total_cost	path
	WAREHOUSE	1	RETAILER	1	30.00	80.00	SUPPLIER-1->WAREHOUSE-1->RETAILER-1

● Identify **bottleneck nodes** — warehouses through which the highest number of shipments flow.

```
-- Identify bottleneck nodes (warehouses with most shipments)
INSERT INTO Shipment (order_id, from_node_type, from_node_id, to_node_type, to_node_id, product_id,
quantity, cost)
VALUES
(1, 'WAREHOUSE', 1, 'RETAILER', 1, 1, 5, 30),
(2, 'WAREHOUSE', 1, 'RETAILER', 2, 1, 3, 25);
SELECT s.from_node_id AS warehouse_id,
       w.warehouse_name,
       COUNT(s.shipment_id) AS shipment_count
FROM Shipment s
JOIN Warehouse w ON s.from_node_type='WAREHOUSE' AND s.from_node_id = w.warehouse_id
GROUP BY s.from_node_id
ORDER BY shipment_count DESC
LIMIT 5;
```

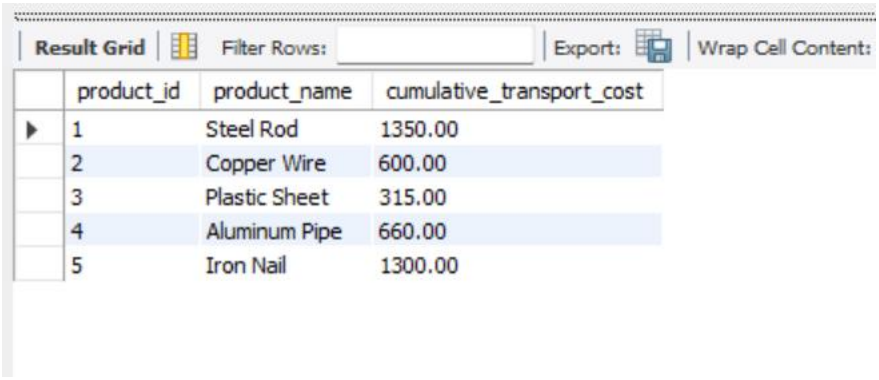
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

warehouse_id	warehouse_name	shipment_count
1	Warehouse W1	3

● Compute **cumulative transportation cost** for each product from supplier → warehouse → retailer.

```
-- Compute cumulative transportation cost for each product from Supplier → Warehouse → Retailer
WITH RECURSIVE ProductRoutes AS (
    SELECT r.route_id, r.from_node_type, r.from_node_id, r.to_node_type, r.to_node_id,
           r.transport_cost AS total_cost
    FROM Route r
    UNION ALL
    SELECT r.route_id, r.from_node_type, r.from_node_id, r.to_node_type, r.to_node_id,
           pr.total_cost + r.transport_cost AS total_cost
    FROM Route r
    INNER JOIN ProductRoutes pr
        ON r.from_node_type = pr.to_node_type
       AND r.from_node_id = pr.to_node_id
)
SELECT sh.product_id, p.product_name,
       SUM(pr.total_cost * sh.quantity) AS cumulative_transport_cost
FROM ProductRoutes pr
JOIN Shipment sh
  ON pr.from_node_type = sh.from_node_type
 AND pr.from_node_id = sh.from_node_id
 AND pr.to_node_type = sh.to_node_type
 AND pr.to_node_id = sh.to_node_id
JOIN Product p ON sh.product_id = p.product_id
```

```
GROUP BY sh.product_id;
```



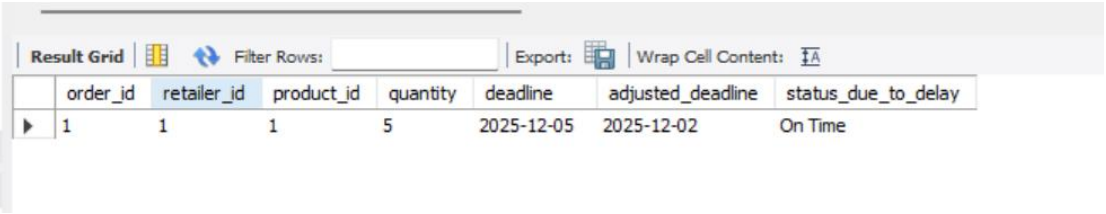
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

product_id	product_name	cumulative_transport_cost
1	Steel Rod	1350.00
2	Copper Wire	600.00
3	Plastic Sheet	315.00
4	Aluminum Pipe	660.00
5	Iron Nail	1300.00

3. Analytical Challenge

● Implement a **what-if analysis** query:
"If supplier S1 is delayed by 3 days, what orders will miss their deadlines?"

```
-- implement a what-if analysis query:
-- Assume supplier 1 is delayed by 3 days
SELECT o.order_id, o.retailer_id, o.product_id, o.quantity, o.deadline,
       DATE_ADD(o.deadline, INTERVAL -3 DAY) AS adjusted_deadline,
       CASE
           WHEN DATE_ADD(CURDATE(), INTERVAL 3 DAY) > o.deadline THEN 'Missed'
           ELSE 'On Time'
       END AS status_due_to_delay
FROM Orders o
JOIN Shipment s ON o.order_id = s.order_id
WHERE s.from_node_type='SUPPLIER' AND s.from_node_id=1;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	retailer_id	product_id	quantity	deadline	adjusted_deadline	status_due_to_delay
1	1	1	5	2025-12-05	2025-12-02	On Time