Day 10 : Global E-commerce Logistics & Supply Chain

Difficulty: Expert+ — multiple strong relationships, multi-leg shipments, inventory valuation, supplier performance, backorders, returns, and time-series analytics.

Goal: Build and analyze a supply-chain dataset covering suppliers → purchase orders → warehouses → inventory → orders → shipments → carriers → customs. Questions focus on complex joins, window functions, recursive CTEs, conditional aggregation, inventory aging, and performance hints.

Scenario description

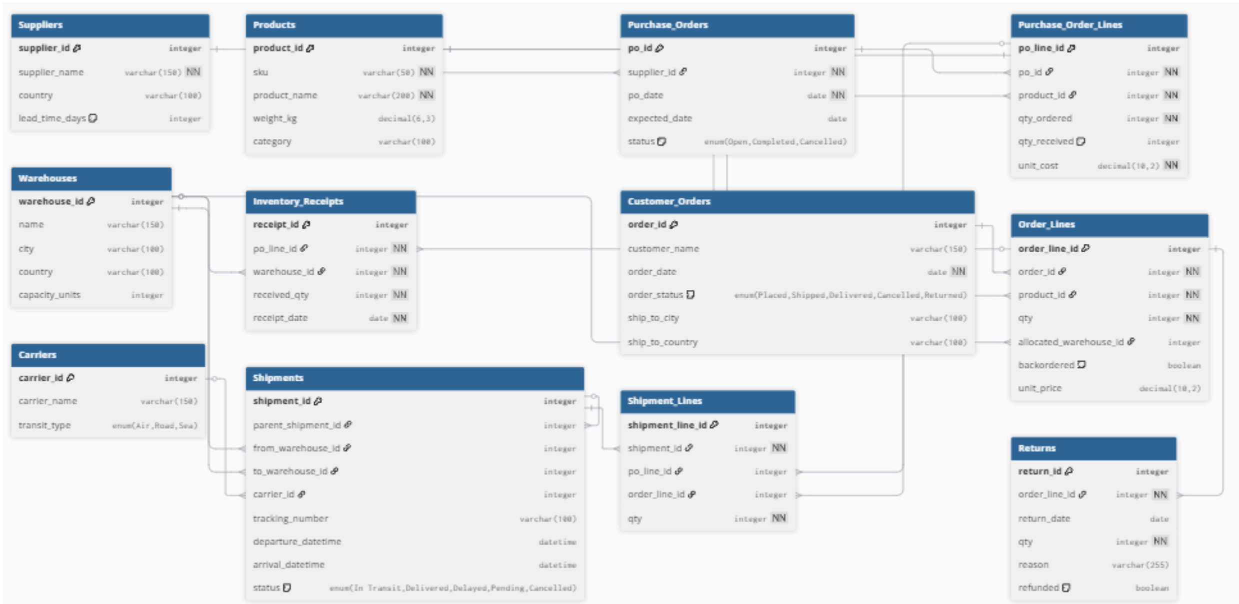
A global e-commerce company manages multiple warehouses across regions, sources products from multiple suppliers, and ships customer orders via multi-leg shipments (warehouse → hub → carrier → destination). Analysts and engineers need to answer operational and strategic questions:

- Where are stockouts and backorders appearing?
- Which suppliers and carriers are underperforming by lead time and delay?
- What's the inventory aging distribution (how long SKUs sit in warehouses)?
- How to compute FIFO cost of goods sold approximations and inventory valuation?
- Track multi-leg shipments and customs delays using recursive route traces.

This dataset intentionally models complexity and real-world edge cases: partial receipts, returns, cancelled POs, backordered quantities, multi-leg shipments with NULL times, and varying statuses.

Database schema (MySQL)

All DDL below is MySQL-compatible. Add indexes as suggested after the schema.



1. Suppliers

```
CREATE TABLE Suppliers (
  supplier_id INT PRIMARY KEY,
  supplier_name VARCHAR(150) NOT NULL,
  country VARCHAR(100),
  lead_time_days INT -- typical lead time
);
```

2. Products

```
CREATE TABLE Products (
  product_id INT PRIMARY KEY,
  sku VARCHAR(50) UNIQUE NOT NULL,
  product_name VARCHAR(200) NOT NULL,
  weight_kg DECIMAL(6,3),
  category VARCHAR(100)
);
```

3. Purchase_Orders

```
CREATE TABLE Purchase_Orders (
  po_id INT PRIMARY KEY,
  supplier_id INT NOT NULL,
  po_date DATE NOT NULL,
  expected_date DATE,
  status ENUM('Open','Completed','Cancelled') DEFAULT 'Open',
  FOREIGN KEY (supplier_id) REFERENCES Suppliers(supplier_id)
);
```

4. Purchase_Order_Lines

```
CREATE TABLE Purchase_Order_Lines (
  po_line_id INT PRIMARY KEY,
  po_id INT NOT NULL,
  product_id INT NOT NULL,
  qty_ordered INT NOT NULL,
  qty_received INT DEFAULT 0,
  unit_cost DECIMAL(10,2) NOT NULL,
  FOREIGN KEY (po_id) REFERENCES Purchase_Orders(po_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

5. Warehouses

```
CREATE TABLE Warehouses (
  warehouse_id INT PRIMARY KEY,
  name VARCHAR(150),
  city VARCHAR(100),
  country VARCHAR(100),
  capacity_units INT
```

```
);
```

6. Inventory_Receipts

(each receipt is a physical receipt to a warehouse; partial receipts allowed)

```
CREATE TABLE Inventory_Receipts (
  receipt_id INT PRIMARY KEY,
  po_line_id INT NOT NULL,
  warehouse_id INT NOT NULL,
  received_qty INT NOT NULL,
  receipt_date DATE NOT NULL,
  FOREIGN KEY (po_line_id) REFERENCES Purchase_Order_Lines(po_line_id),
  FOREIGN KEY (warehouse_id) REFERENCES Warehouses(warehouse_id)
);
```

7. Customer_Orders

```
CREATE TABLE Customer_Orders (
  order_id INT PRIMARY KEY,
  customer_name VARCHAR(150),
  order_date DATE NOT NULL,
  order_status ENUM('Placed','Shipped','Delivered','Cancelled','Returned') DEF
AULT 'Placed',
  ship_to_city VARCHAR(100),
  ship_to_country VARCHAR(100)
);
```

8. Order_Lines

```
CREATE TABLE Order_Lines (
  order_line_id INT PRIMARY KEY,
```

```
  order_id INT NOT NULL,
  product_id INT NOT NULL,
  qty INT NOT NULL,
  allocated_warehouse_id INT NULL, -- which warehouse will fulfill
  backordered BOOLEAN DEFAULT FALSE,
  unit_price DECIMAL(10,2),
  FOREIGN KEY (order_id) REFERENCES Customer_Orders(order_id),
  FOREIGN KEY (product_id) REFERENCES Products(product_id),
  FOREIGN KEY (allocated_warehouse_id) REFERENCES Warehouses(warehou
se_id)
);
```

9. Carriers

```
CREATE TABLE Carriers (
  carrier_id INT PRIMARY KEY,
  carrier_name VARCHAR(150),
  transit_type ENUM('Air','Road','Sea')
);
```

10. Shipments

(Multi-leg shipments; `parent_shipment_id` allows building a route tree)

```
CREATE TABLE Shipments (
  shipment_id INT PRIMARY KEY,
  parent_shipment_id INT NULL,
  from_warehouse_id INT NULL,
  to_warehouse_id INT NULL,
  carrier_id INT NULL,
  tracking_number VARCHAR(100),
  departure_datetime DATETIME,
  arrival_datetime DATETIME,
```

```
  status ENUM('In Transit','Delivered','Delayed','Pending','Cancelled') DEFAULT
'Pending',
  FOREIGN KEY (parent_shipment_id) REFERENCES Shipments(shipment_id),
  FOREIGN KEY (from_warehouse_id) REFERENCES Warehouses(warehouse_i
d),
  FOREIGN KEY (to_warehouse_id) REFERENCES Warehouses(warehouse_id),
  FOREIGN KEY (carrier_id) REFERENCES Carriers(carrier_id)
);
```

11. Shipment_Lines

(associates order lines or receipts with shipments)

```
CREATE TABLE Shipment_Lines (
  shipment_line_id INT PRIMARY KEY,
  shipment_id INT NOT NULL,
  po_line_id INT NULL,
  order_line_id INT NULL,
  qty INT NOT NULL,
  FOREIGN KEY (shipment_id) REFERENCES Shipments(shipment_id),
  FOREIGN KEY (po_line_id) REFERENCES Purchase_Order_Lines(po_line_id),
  FOREIGN KEY (order_line_id) REFERENCES Order_Lines(order_line_id)
);
```

12. Returns

```
CREATE TABLE Returns (
  return_id INT PRIMARY KEY,
  order_line_id INT NOT NULL,
  return_date DATE,
  qty INT NOT NULL,
  reason VARCHAR(255),
  refunded BOOLEAN DEFAULT FALSE,
```

```
  FOREIGN KEY (order_line_id) REFERENCES Order_Lines(order_line_id)
);
```

Sample data (representative, covers edge cases: partial receipts, backorders, NULL datetimes)

```
-- Suppliers
INSERT INTO Suppliers VALUES
(1,'Alpha Foods','India',10),
(2,'Global Imports','China',25),
(3,'Local Farms','India',5);

-- Products
INSERT INTO Products VALUES
(100,'SKU-PA-01','Paneer Age-ture',0.6,'Dairy'),
(101,'SKU-PZ-02','Pizza Base',0.4,'Bakery'),
(102,'SKU-SS-05','Sesame Seeds',0.02,'Grocery'),
(103,'SKU-BX-09','Box Packaging',0.5,'Packaging');

-- Warehouses
INSERT INTO Warehouses VALUES
(10,'Mumbai WH1','Mumbai','India',100000),
(11,'Delhi WH1','Delhi','India',80000),
(12,'Shanghai Hub','Shanghai','China',200000);

-- Purchase Orders and lines (one PO partially received; one cancelled)
INSERT INTO Purchase_Orders VALUES
(500,'Alpha Foods', '2024-09-01', '2024-09-11', 'Completed'), -- note: supplier_id numeric expected; we'll use correct values below
(501,2,'2024-09-02','2024-09-27','Open'),
(502,3,'2024-09-10','2024-09-15','Cancelled');

-- Fixing PO inserts properly
```

```
DELETE FROM Purchase_Orders;
INSERT INTO Purchase_Orders VALUES
(500,1,'2024-09-01','2024-09-11','Completed'),
(501,2,'2024-09-02','2024-09-27','Open'),
(502,3,'2024-09-10','2024-09-15','Cancelled');

INSERT INTO Purchase_Order_Lines VALUES
(5001,500,100,1000,800,50.00), -- ordered 1000, received 800
(5002,500,103,500,500,2.00),   -- fully received
(5003,501,101,2000,0,30.00),   -- open, not yet received
(5004,502,102,10000,0,0.10);    -- cancelled PO line

-- Inventory receipts (partial receives; one to Shanghai hub)
INSERT INTO Inventory_Receipts VALUES
(9001,5001,10,500,'2024-09-05'),
(9002,5001,11,300,'2024-09-07'),
(9003,5002,10,500,'2024-09-06'),
(9004,5003,12,0,'2024-09-20'); -- none received yet

-- Customer orders (some allocated, some backordered)
INSERT INTO Customer_Orders VALUES
(2000,'John Buyer','2024-09-08','Placed','Mumbai','India'),
(2001,'Sara Buyer','2024-09-09','Placed','Delhi','India');

INSERT INTO Order_Lines VALUES
(7001,2000,100,200,10,FALSE,120.00), -- allocated to Mumbai WH1 later
(7002,2000,101,100,11,FALSE,50.00),  -- allocated to Delhi
(7003,2001,100,100, NULL, TRUE,120.00); -- backordered (no allocation)

-- Carriers
INSERT INTO Carriers VALUES
(300, 'FastAir', 'Air'),
(301, 'OceanicLines', 'Sea'),
(302, 'RoadExpress', 'Road');

-- Shipments (multi-leg: PO line parts shipped from supplier to Shanghai hub t
```

```
hen to Mumbai)
INSERT INTO Shipments VALUES
(8000,NULL, NULL,12,300,'TRK-PO-5001-LEG1','2024-09-03 08:00:00','2024-
09-10 18:00:00','Delivered'),
(8001,8000,12,10,300,'TRK-PO-5001-LEG2','2024-09-12 06:00:00','2024-09-1
5 09:00:00','Delivered'),
(8002,NULL,11,10,302,'TRK-PO-5001-DOM','2024-09-06 09:00:00',NULL,'In Tr
ansit'); -- missing arrival_datetime

INSERT INTO Shipment_Lines VALUES
(8100,8000,5001,NULL,500),
(8101,8001,5001,NULL,300),
(8102,8002,5002,NULL,500),
(8103,8002,NULL,7001,200); -- shipping order line 7001 qty 200

-- Returns (partial return)
INSERT INTO Returns VALUES
(4001,7001,'2024-09-20',20,'Damaged product',TRUE);
```

> Note: sample data covers: partial receipts (qty_received less than qty_ordered), backordered order lines (allocated_warehouse_id NULL + backordered TRUE), multi-leg shipments (parent_shipment_id chain), shipments with NULL arrival_datetime (in transit), and cancelled PO.

ERD (textual / ASCII)

```
Suppliers (1) ──< Purchase_Orders (M) ──< Purchase_Order_Lines (M) ──< I
nventory_Receipts (M)
                                        \
                                         └─< Shipment_Lines (M) >─ Shipments (1) ──
< Shipments (parent-child multi-leg)
Products (1) ──< Purchase_Order_Lines (M)
Products (1) ──< Order_Lines (M) ──< Customer_Orders (1)
```

```
Warehouses (1) ──< Inventory_Receipts (M)
Warehouses (1) ──< Shipments (as from_warehouse / to_warehouse)
Carriers (1) ──< Shipments (M)
Order_Lines (1) ──< Shipment_Lines (M)
Order_Lines (1) ──< Returns (M)
```

Cardinalities: typical 1:M everywhere; shipments can be chained (multi-leg) via `parent_shipment_id` (1:M).

Questions (5) — increased complexity

Each question uses MySQL syntax only.

Easy (but tricky): Q1 — Current on-hand quantity per product per warehouse

Compute current on-hand quantity for each `product_id` and `warehouse_id` as:

```
SUM(received_qty from Inventory_Receipts to that warehouse for po_line.product)
- SUM(qty shipped out for that product from that warehouse via Shipment_Lines)
- SUM(qty returned back into warehouse?) (for simplicity returns are NOT restocked here)
```

Return rows with zero or negative values (to highlight data issues). Include product SKU and warehouse name.

Medium: Q2 — Backorder report

List all order lines currently backordered (Order_Lines.backordered = TRUE), show:

- order_line_id, order_id, product sku, qty, earliest open PO expected_date that can fulfill remaining qty (i.e., PO with status 'Open' and qty_remaining > 0), and days_until_expected (expected_date - today). If none found, show NULLs.

Assume `qty_remaining` on a PO line = qty_ordered - qty_received.

Hard: Q3 — Supplier on-time performance

For each supplier compute:

- total_po_count,
- on_time_count (POs where all lines were received and actual last receipt date <= expected_date),
- pct_on_time (on_time_count / total_po_count*100).

Consider only PO.status IN ('Completed','Open') where expected_date IS NOT NULL. Use subqueries/CTEs.

Difficult: Q4 — Multi-leg shipment route width & delays

Using a recursive CTE, for each top-level shipment (shipment with parent_shipment_id IS NULL), compute:

- shipment_id,
- total_legs (count of nodes in its route),
- total_transit_time_minutes (sum of minutes between departure_datetime and arrival_datetime across legs where both datetimes exist),
- max_leg_delay_minutes (maximum difference between scheduled (not stored) and actual arrival — since scheduled unknown, instead compute any leg with arrival_datetime IS NULL counted as 'open' delay and treat their gap as NOW() - departure_datetime in minutes),
- status (if any leg status = 'Delayed' or 'In Transit' then 'Problematic' else 'OK').

Return top 5 problematic shipments ordered by max_leg_delay_minutes DESC.

Expert: Q5 — FIFO-style approximate COGS & Inventory Valuation

Approximate FIFO valuation for product_id = 100 at warehouse_id = 10:

- Use `Inventory_Receipts` ordered by `receipt_date` ascending as the FIFO "layers" with `received_qty` and `unit_cost` from the PO line.
- Compute valuation of current on-hand quantity at warehouse (i.e., how much value remains using earliest receipts consumed first by shipments associated with that warehouse).
- Also compute total COGS for shipments of that product from that warehouse (i.e., qty shipped * unit_cost of layers consumed).

Return:

- warehouse_id, product_id, on_hand_qty, on_hand_value, total_shipped_qty, total_cogs

If insufficient data, explain assumptions inline in the answer.

Solutions (MySQL queries + explanations & optimization tips)

> Notes before queries:
>
> - MySQL supports CTEs (including recursive) since 8.0.
> - Some queries use `LEFT JOIN` to show NULLs.
> - For performance on real systems add indexes suggested after each query.

Q1 — On-hand qty per product per warehouse

```
-- Q1: On-hand per product per warehouse
SELECT
  p.product_id,
  p.sku,
  w.warehouse_id,
  w.name AS warehouse_name,
  COALESCE(SUM(ir.received_qty),0) AS total_received,
  COALESCE(SUM(sl_out.qty),0)      AS total_shipped_out,
  (COALESCE(SUM(ir.received_qty),0) - COALESCE(SUM(sl_out.qty),0)) AS on
```

```
_hand_qty
FROM Products p
CROSS JOIN Warehouses w
LEFT JOIN Purchase_Order_Lines pol ON pol.product_id = p.product_id
LEFT JOIN Inventory_Receipts ir ON ir.po_line_id = pol.po_line_id AND ir.wareh
ouse_id = w.warehouse_id
LEFT JOIN (
   SELECT sl.shipment_id, sl.po_line_id, sl.order_line_id, sl.qty, s.from_wareho
use_id
   FROM Shipment_Lines sl
   JOIN Shipments s ON sl.shipment_id = s.shipment_id
) AS sl_out ON sl_out.po_line_id = pol.po_line_id AND sl_out.from_warehouse_i
d = w.warehouse_id
GROUP BY p.product_id, p.sku, w.warehouse_id, w.name
HAVING on_hand_qty IS NOT NULL
ORDER BY p.product_id, w.warehouse_id;
```

Explanation & reasoning:

- We cross-join `Products` × `Warehouses` to show every combination (so zero and negative values show).
- `Inventory_Receipts` gives received quantities into each warehouse for each PO line. Sum them.
- `Shipment_Lines` joined to `Shipments` gives outgoing shipments from warehouses (`from_warehouse_id`). We subtract shipped qty.
- Returns negative values if more shipped than received — useful to detect data problems.

Index suggestions:

- `Inventory_Receipts(po_line_id, warehouse_id, receipt_date)`
- `Shipment_Lines(po_line_id)`, `Shipments(shipment_id, from_warehouse_id)`

Q2 — Backorder report with earliest open PO expected_date

```
-- Q2: Backorder report
WITH po_remaining AS (
  SELECT
    pol.po_line_id,
    pol.product_id,
    po.supplier_id,
    po.expected_date,
    (pol.qty_ordered - COALESCE(pol.qty_received,0)) AS qty_remaining,
    po.status
  FROM Purchase_Order_Lines pol
  JOIN Purchase_Orders po ON pol.po_id = po.po_id
  WHERE po.status = 'Open' AND (pol.qty_ordered - COALESCE(pol.qty_receiv
ed,0)) > 0
)
SELECT
  ol.order_line_id,
  ol.order_id,
  p.sku,
  ol.qty AS order_qty,
  pr.po_line_id AS candidate_po_line_id,
  pr.expected_date,
  DATEDIFF(pr.expected_date, CURDATE()) AS days_until_expected,
  pr.qty_remaining
FROM Order_Lines ol
JOIN Products p ON ol.product_id = p.product_id
LEFT JOIN (
  SELECT pr1.* FROM po_remaining pr1
  JOIN (
    -- earliest expected_date per product
    SELECT product_id, MIN(expected_date) AS min_expected
    FROM po_remaining
    GROUP BY product_id
  ) prmin ON pr1.product_id = prmin.product_id AND pr1.expected_date = prmi
n.min_expected
) pr ON pr.product_id = ol.product_id
```

```
WHERE ol.backordered = TRUE
ORDER BY ol.order_line_id;
```

Explanation:

- Compute `po_remaining` CTE to find open PO lines with remaining qty.
- For each backordered `Order_Lines` find the PO with the earliest `expected_date` for that product.
- `DATEDIFF` gives days until expected (can be negative if expected_date in past).
- If no PO exists, `candidate_po_line_id` and `expected_date` will be NULL.

Assumptions:

- A single earliest PO is used to indicate nearest fulfillment source; real systems may need to allocate across multiple POs.

Index suggestions:

- `Purchase_Order_Lines(product_id, po_id)`
- `Purchase_Orders(status, expected_date)`

Q3 — Supplier on-time performance

```
-- Q3: Supplier on-time performance
WITH po_last_receipt AS (
  SELECT
    po.po_id,
    po.supplier_id,
    po.expected_date,
    po.status,
    MAX(ir.receipt_date) AS last_receipt_date,
    -- sum lines to check fully received
    SUM(pol.qty_ordered) AS po_qty_ordered,
    SUM(pol.qty_received) AS po_qty_received
  FROM Purchase_Orders po
  LEFT JOIN Purchase_Order_Lines pol ON po.po_id = pol.po_id
```

```
  LEFT JOIN Inventory_Receipts ir ON pol.po_line_id = ir.po_line_id
  WHERE po.expected_date IS NOT NULL AND po.status IN ('Open','Complete
d')
  GROUP BY po.po_id, po.supplier_id, po.expected_date, po.status
)
SELECT
  s.supplier_id,
  s.supplier_name,
  COUNT(pl.po_id) AS total_po_count,
  SUM(CASE WHEN (pl.po_qty_ordered = pl.po_qty_received) AND pl.last_rece
ipt_date <= pl.expected_date THEN 1 ELSE 0 END) AS on_time_count,
  ROUND(100 * SUM(CASE WHEN (pl.po_qty_ordered = pl.po_qty_received) A
ND pl.last_receipt_date <= pl.expected_date THEN 1 ELSE 0 END) / COUNT(pl.
po_id), 2) AS pct_on_time
FROM Suppliers s
LEFT JOIN po_last_receipt pl ON pl.supplier_id = s.supplier_id
GROUP BY s.supplier_id, s.supplier_name
ORDER BY pct_on_time DESC;
```

Explanation:

- `po_last_receipt` aggregates per PO: total ordered vs total received and last receipt date.
- PO counted as on-time if fully received and last receipt date <= expected_date.
- Then aggregate per supplier.

Edge cases handled:

- `LEFT JOIN` allows suppliers with no POs to appear with zero totals.
- POs with partial receipts are not counted as on-time.

Index suggestions:

- `Purchase_Orders(supplier_id, expected_date)`
- `Purchase_Order_Lines(po_id, product_id)`

- `Inventory_Receipts(po_line_id, receipt_date)`

Q4 — Multi-leg shipment route analysis (recursive CTE)

```
-- Q4: Multi-leg shipment analysis
WITH RECURSIVE route AS (
  -- base: top-level shipments (parent_shipment_id IS NULL)
  SELECT
    s.shipment_id,
    s.shipment_id AS root_shipment_id,
    s.parent_shipment_id,
    s.departure_datetime,
    s.arrival_datetime,
    s.status,
    TIMESTAMPDIFF(MINUTE, s.departure_datetime, s.arrival_datetime) AS leg
_minutes,
    CASE WHEN s.arrival_datetime IS NULL THEN TIMESTAMPDIFF(MINUTE, s.
departure_datetime, NOW()) ELSE 0 END AS open_leg_minutes,
    1 AS leg_seq
  FROM Shipments s
  WHERE s.parent_shipment_id IS NULL

  UNION ALL

  -- recursive: child legs
  SELECT
    c.shipment_id,
    r.root_shipment_id,
    c.parent_shipment_id,
    c.departure_datetime,
    c.arrival_datetime,
    c.status,
    TIMESTAMPDIFF(MINUTE, c.departure_datetime, c.arrival_datetime) AS leg
_minutes,
    CASE WHEN c.arrival_datetime IS NULL THEN TIMESTAMPDIFF(MINUTE, c.
```

```
departure_datetime, NOW()) ELSE 0 END AS open_leg_minutes,
    r.leg_seq + 1
  FROM Shipments c
  JOIN route r ON c.parent_shipment_id = r.shipment_id
)
SELECT
  root_shipment_id AS top_shipment_id,
  COUNT(*) AS total_legs,
  SUM(COALESCE(leg_minutes,0)) AS total_transit_time_minutes,
  GREATEST(MAX(open_leg_minutes), 0) AS max_leg_delay_minutes,
  CASE WHEN SUM(CASE WHEN status IN ('Delayed','In Transit') THEN 1 ELSE
0 END) > 0 THEN 'Problematic' ELSE 'OK' END AS overall_status
FROM route
GROUP BY root_shipment_id
HAVING overall_status = 'Problematic'
ORDER BY max_leg_delay_minutes DESC
LIMIT 5;
```

Explanation:

- CTE `route` recursively traverses shipment legs starting from root shipments (`parent_shipment_id IS NULL`).
- For each leg compute `leg_minutes` when arrival exists; otherwise treat open leg delay as `NOW() - departure_datetime`.
- Group by root shipment to compute total legs, sum of transit minutes, and maximum open leg delay.
- Return top problematic shipments.

Index suggestions:

- `Shipments(parent_shipment_id)` for recursion traversal.
- `Shipments(departure_datetime, arrival_datetime, status)` for time calculations.

Q5 — FIFO-style approximate COGS & inventory valuation for product 100 at warehouse 10

Assumptions & approach:

- Use `Inventory_Receipts` for warehouse 10 and product 100, ordered by `receipt_date` as FIFO layers.
- Each purchase receipt is linked to a PO line; use `unit_cost` from that PO line.
- Determine total received_qty (sum receipts), total shipped_qty (sum of Shipment_Lines where shipment.from_warehouse_id = 10 and shipment_line.po_line_id corresponds to product 100 OR where shipment_line.order_line_id references Order_Lines for that product and from_warehouse_id = 10). For this simplified sample we consider shipments that reference `po_line_id` and shipments that reference `order_line_id` (which relate to product_id via order_lines).
- Compute consumption of layers in FIFO order: earliest receipts are consumed first to meet total_shipped_qty; remaining quantity across layers = on_hand_qty.

This requires iterative consumption; emulate with windowed cumulative sums.

```
-- Q5: FIFO valuation for product_id=100 at warehouse_id=10

-- Step 1: build receipt layers with unit_cost
WITH receipt_layers AS (
  SELECT
    ir.receipt_id,
    pol.po_line_id,
    pol.product_id,
    ir.warehouse_id,
    ir.received_qty,
    pol.unit_cost,
    ir.receipt_date
  FROM Inventory_Receipts ir
  JOIN Purchase_Order_Lines pol ON ir.po_line_id = pol.po_line_id
  WHERE pol.product_id = 100 AND ir.warehouse_id = 10
```

```
  ORDER BY ir.receipt_date
),

-- Step 2: compute total shipped qty of product 100 from warehouse 10
shipped AS (
  SELECT COALESCE(SUM(sl.qty),0) AS total_shipped_qty
  FROM Shipment_Lines sl
  JOIN Shipments s ON sl.shipment_id = s.shipment_id
  LEFT JOIN Purchase_Order_Lines pol ON sl.po_line_id = pol.po_line_id
  LEFT JOIN Order_Lines ol ON sl.order_line_id = ol.order_line_id
  WHERE s.from_warehouse_id = 10
    AND ( (pol.product_id = 100) OR (ol.product_id = 100) )
),

-- Step 3: annotate receipt layers with cumulative received and compute rema
ining after shipments
layer_cum AS (
  SELECT
    rl.*,
    SUM(rl.received_qty) OVER (ORDER BY rl.receipt_date, rl.receipt_id ROWS
BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cum_received
  FROM receipt_layers rl
),

-- Step 4: compute layer consumption using shipped qty
consumption AS (
  SELECT
    lc.*,
    s.total_shipped_qty,
    GREATEST(0, LEAST(lc.cum_received, s.total_shipped_qty) - COALESCE(
      LAG(lc.cum_received) OVER (ORDER BY lc.receipt_date, lc.receipt_id), 0
    )) AS consumed_from_layer
  FROM layer_cum lc
  CROSS JOIN shipped s
)
```

```
SELECT
  10 AS warehouse_id,
  100 AS product_id,
  COALESCE(SUM(consumed_from_layer),0) AS total_shipped_qty_computed,
  COALESCE(SUM(consumed_from_layer * unit_cost),0.00) AS total_cogs,
  COALESCE(SUM(received_qty) - SUM(consumed_from_layer),0) AS on_hand
_qty,
  COALESCE(SUM( (received_qty - consumed_from_layer) * unit_cost ),0.00)
AS on_hand_value
FROM consumption;
```

Explanation (step-by-step):

1. `receipt_layers` lists receipts of product 100 into warehouse 10 with `unit_cost` (from associated PO line).
2. `shipped` calculates total shipped quantity of that product from warehouse 10 (via `Shipment_Lines` joined to `Shipments` and related `po_line_id` or `order_line_id`).
3. `layer_cum` computes cumulative receipts ordered by `receipt_date` — representing FIFO layers.
4. `consumption` uses the `total_shipped_qty` and cumulative receipt to compute how much of each layer is consumed:
   - For each layer, consumed = min(cum_received, total_shipped) – previous_cum_received (clamped to >=0).
5. Final SELECT aggregates consumed amounts to produce `total_cogs` (sum(consumed_from_layer * unit_cost)), `on_hand_qty` (sum(received - consumed)), and `on_hand_value` (value of remaining units at layer costs).

Assumptions & caveats:

- This is an approximation; real FIFO COGS needs to handle partial shipments mapped to specific receipts, returns restocking, and inter-warehouse transfers.
- `unit_cost` taken from `Purchase_Order_Lines` is assumed to reflect the cost of the receipt.

- Shipments that reference `order_line_id` consume stock; shipments that reference `po_line_id` move receipts between warehouses — both considered in shipped calculation depending on business rules.

Index suggestions:

- `Inventory_Receipts(po_line_id, warehouse_id, receipt_date)`
- `Purchase_Order_Lines(po_line_id, product_id, unit_cost)`
- `Shipment_Lines(shipment_id, po_line_id, order_line_id)`
- `Shipments(shipment_id, from_warehouse_id, departure_datetime)`

General optimization & data-quality tips (production)

1. **Indexing:** create composite indexes on join columns (e.g., `Purchase_Order_Lines(po_id, product_id)`, `Inventory_Receipts(po_line_id, warehouse_id)`, `Shipment_Lines(shipment_id, po_line_id, order_line_id)`).
2. **Partitioning:** partition large tables by date (e.g., `Inventory_Receipts(receipt_date)` and `Shipments(departure_datetime)`).
3. **Materialized summaries:** maintain daily aggregates for `received`, `shipped`, and `on_hand` (ETL jobs) to speed reporting.
4. **Normalization vs Denormalization:** keep normalized source tables but maintain denormalized snapshot tables for analytics.
5. **Handle NULL datetimes:** treat `NULL arrival_datetime` as `in transit`; use `NOW()` with caution (non-deterministic).
6. **Data integrity checks:** scheduled jobs to check `qty_shipped <= qty_received` per warehouse/product, and flag inconsistencies.