To help you practice **Microsoft SQL Server (T-SQL)** using your datasets, I have designed four practical exercises. These range from standard data retrieval to advanced logical structures.

1. Subqueries & Joins

Scenario: Analyze the fulfillment efficiency of your Purchase Orders (LPOs). You want to see how much was issued compared to what was originally delivered for the same LPO.

Task: Write a query that joins commodities_delivered (D) and commodities_issued (I) on LPO_Number. Calculate the "Issue Percentage" (IssuedQuantity / QtyOnHand). Use a **subquery** in the WHERE clause to only include Suppliers who have a ProcurementFee higher than the average fee.

SQL

```
SELECT
    D.LPO_Number,
    D.ItemCode AS DeliveredItem,
    I.ItemCode AS IssuedItem,
    D.QtyOnHand AS DeliveredQty,
    I.IssuedQuantity,
    (CAST(I.IssuedQuantity AS FLOAT) / NULLIF(D.QtyOnHand, 0)) * 100 AS IssuePercentage,
    D.SupplierName
FROM commodities_delivered D
INNER JOIN commodities_issued I ON D.LPO_Number = I.LPO_Number
WHERE D.SupplierID IN (
    SELECT SupplierID
    FROM commodities_delivered
    WHERE ProcurementFee > (SELECT AVG(ProcurementFee) FROM commodities_delivered)
);
```

---

2. Window Functions

Scenario: Performance Ranking. You want to see which Warehouses are handling the most

value and how they compare to the overall organization.

Task: For each warehouse, calculate the total TotalSales. Use RANK() to rank warehouses by sales and use a windowed SUM() to show the "Running Total" of sales across all warehouses.

SQL

```
SELECT
    WarehouseID,
    WarehouseLocation,
    TotalWarehouseSales,
    RANK() OVER (ORDER BY TotalWarehouseSales DESC) AS SalesRank,
    SUM(TotalWarehouseSales) OVER (ORDER BY TotalWarehouseSales DESC) AS RunningTotal,
    CAST(TotalWarehouseSales * 100.0 / SUM(TotalWarehouseSales) OVER() AS DECIMAL(10,2))
AS PctOfTotal
FROM (
    SELECT
        WarehouseID,
        WarehouseLocation,
        SUM(TotalSales) AS TotalWarehouseSales
    FROM commodities_delivered
    GROUP BY WarehouseID, WarehouseLocation
) AS WarehouseSummary;
```

---

3. Query Optimization

Scenario: Selecting the "Top N" per group. You need to find the single most expensive item (highest UnitCost) delivered for every Level1Category.

Task: Compare two ways of doing this. The ROW_NUMBER() method is generally the **optimized** standard in SQL Server because it avoids multiple scans of the same table.

Optimized Version (using ROW_NUMBER):

SQL

```sql
WITH RankedItems AS (
    SELECT
        Level1Category,
        ItemCode,
        UnitCost,
        SupplierName,
        ROW_NUMBER() OVER (PARTITION BY Level1Category ORDER BY UnitCost DESC) AS CostRank
    FROM commodities_delivered
)
SELECT * FROM RankedItems
WHERE CostRank = 1;
```

Note: This is faster than using a Correlated Subquery with MAX(), as it only requires one pass over the data.

---

4. Recursive CTEs

Scenario: Supply Chain Simulation. Since the data doesn't have a natural parent-child hierarchy (like an Org Chart), we can use a Recursive CTE to generate a simulated "Tracking Timeline" for an item's journey.

Task: Create a recursive CTE that starts with a specific ReceiptDate and generates 5 subsequent "Audit Checkpoints" every 30 days to simulate a maintenance or inspection schedule.

SQL

```sql
WITH MaintenanceSchedule AS (
    -- Anchor Member: Start with a specific receipt
    SELECT
        ReceiptID,
        ItemCode,
        ReceiptDate AS CheckpointDate,
        1 AS CheckpointNumber
    FROM commodities_delivered
    WHERE ReceiptID = 'RCPT-000001'

    UNION ALL
```

```
    -- Recursive Member: Add 30 days for the next 5 checkpoints
    SELECT
        ReceiptID,
        ItemCode,
        DATEADD(day, 30, CheckpointDate),
        CheckpointNumber + 1
    FROM MaintenanceSchedule
    WHERE CheckpointNumber < 5
)
SELECT * FROM MaintenanceSchedule;
```

Quick Tips for SQL Server:

- **Column Aliases:** Always use AS for readability.
- **Data Types:** Be careful with integer division; use CAST(... AS FLOAT) when calculating percentages.
- **Performance:** In the "Optimization" query, look at the **Execution Plan** (Ctrl+L in SSMS) to see how the Window Function compares to a subquery.