**Case Study 1: Recursive Queries for Hierarchical Data**

**Scenario:**  
A company stores its employee hierarchy in a table with columns employee\_id, name, and manager\_id. Write a query to retrieve the full reporting hierarchy of a specific employee.

**Solution Approach:**  
Use a recursive Common Table Expression (CTE) to iterate through the hierarchy.

WITH RECURSIVE EmployeeHierarchy AS (

SELECT employee\_id, name, manager\_id, 1 AS level

FROM employees

WHERE employee\_id = :target\_id

UNION ALL

SELECT e.employee\_id, e.name, e.manager\_id, eh.level + 1

FROM employees e

INNER JOIN EmployeeHierarchy eh ON e.manager\_id = eh.employee\_id

)

SELECT \* FROM EmployeeHierarchy;

**Case Study 2: Dynamic Pivot Tables**

**Scenario:**  
Given a table sales with columns region, month, and revenue, generate a pivot table to show regions as rows and months as columns with revenue as the values.

**Solution Approach:**  
Use dynamic SQL with CASE or native pivoting features.

SELECT region,

SUM(CASE WHEN month = 'January' THEN revenue ELSE 0 END) AS January,

SUM(CASE WHEN month = 'February' THEN revenue ELSE 0 END) AS February

FROM sales

GROUP BY region;

**Case Study 3: Ranking Products by Sales**

**Scenario:**  
A sales table contains columns product\_id, region, and revenue. Find the top 3 products by revenue for each region.

**Solution Approach:**  
Use the ROW\_NUMBER() function with a PARTITION BY clause.

WITH RankedProducts AS (

SELECT product\_id, region, revenue,

ROW\_NUMBER() OVER (PARTITION BY region ORDER BY revenue DESC) AS rank

FROM sales

)

SELECT \* FROM RankedProducts WHERE rank <= 3;

**Case Study 4: Detecting Data Gaps**

**Scenario:**  
A timestamps table logs user actions with a user\_id and action\_time. Find gaps in action times greater than 1 hour for each user.

**Solution Approach:**  
Use LAG() to compare timestamps row-by-row.

WITH ActionGaps AS (

SELECT user\_id, action\_time,

LAG(action\_time) OVER (PARTITION BY user\_id ORDER BY action\_time) AS prev\_action\_time

FROM timestamps

)

SELECT user\_id, prev\_action\_time, action\_time

FROM ActionGaps

WHERE action\_time - prev\_action\_time > INTERVAL '1 hour';

**Case Study 5: Window Functions for Moving Averages**

**Scenario:**  
Given a sales table with date, region, and revenue, calculate a 7-day moving average of revenue for each region.

**Solution Approach:**  
Use the WINDOW clause with ROWS BETWEEN.

SELECT date, region, revenue,

AVG(revenue) OVER (PARTITION BY region ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS moving\_avg

FROM sales;

**Case Study 6: Time Series Analysis with Gaps**

**Scenario:**  
Fill in missing dates in a sales table with zero revenue for each region.

**Solution Approach:**  
Use a calendar table or generate dates dynamically and join with the sales data.

WITH Calendar AS (

SELECT generate\_series(MIN(date), MAX(date), INTERVAL '1 day') AS calendar\_date

FROM sales

)

SELECT c.calendar\_date, s.region, COALESCE(s.revenue, 0) AS revenue

FROM Calendar c

CROSS JOIN (SELECT DISTINCT region FROM sales) r

LEFT JOIN sales s ON c.calendar\_date = s.date AND r.region = s.region;

**Case Study 7: Advanced Joins with Fuzzy Matching**

**Scenario:**  
Match customers between two datasets (customers\_a and customers\_b) where names may differ slightly due to typos.

**Solution Approach:**  
Use string similarity functions like LEVENSHTEIN or SOUNDEX.

SELECT a.customer\_id, b.customer\_id, LEVENSHTEIN(a.name, b.name) AS similarity\_score

FROM customers\_a a

JOIN customers\_b b ON LEVENSHTEIN(a.name, b.name) < 3;

**Case Study 8: Complex Subqueries**

**Scenario:**  
A projects table has columns project\_id and completion\_time. Find the projects whose completion time is above the average for their category.

**Solution Approach:**  
Use correlated subqueries.

SELECT project\_id, completion\_time

FROM projects p1

WHERE completion\_time > (

SELECT AVG(completion\_time)

FROM projects p2

WHERE p1.category = p2.category

);

**Case Study 9: Cumulative Totals and Percentages**

**Scenario:**  
For a sales table with date, region, and revenue, calculate the cumulative revenue and its percentage of total revenue.

**Solution Approach:**  
Use window functions for cumulative sums.

WITH TotalRevenue AS (SELECT SUM(revenue) AS total FROM sales)

SELECT date, region, revenue,

SUM(revenue) OVER (PARTITION BY region ORDER BY date) AS cumulative\_revenue,

SUM(revenue) OVER (PARTITION BY region ORDER BY date) \* 100.0 / (SELECT total FROM TotalRevenue) AS percentage

FROM sales;

**Case Study 10: Analyzing Clickstream Data**

**Scenario:**  
Analyze a clickstream table to find the most common paths users take across webpages.

**Solution Approach:**  
Use LAG() and LEAD() functions to track paths.

WITH UserPaths AS (

SELECT user\_id, page,

LAG(page) OVER (PARTITION BY user\_id ORDER BY timestamp) AS previous\_page

FROM clickstream

)

SELECT previous\_page, page, COUNT(\*) AS path\_count

FROM UserPaths

GROUP BY previous\_page, page

ORDER BY path\_count DESC;