1. Second highest salary

- SELECT MAX(salary)
- FROM employees
- WHERE salary < (SELECT MAX(salary) FROM employees);

- ✅ Uses subquery to exclude the top salary.

2. Department-wise avg salary (with >5 employees)

- SELECT department_id, AVG(salary) AS avg_salary
- FROM employees
- GROUP BY department_id
- HAVING COUNT(*) > 5;

- ✅ `HAVING` filters aggregated groups.

3. Customers with >3 orders in last 30 days

- SELECT customer_id
- FROM orders
- WHERE order_date >= CURRENT_DATE - INTERVAL '30 days'
- GROUP BY customer_id
- HAVING COUNT(*) > 3;

- ✅ Aggregation + date filter.

4. Find duplicates on multiple fields

- SELECT col1, col2, COUNT(*)
- FROM my_table
- GROUP BY col1, col2
- HAVING COUNT(*) > 1;

- ✅ Helps detect data quality issues.

5. Transpose rows into columns using CASE

- SELECT
- user_id,
- MAX(CASE WHEN month = 'Jan' THEN spend END) AS Jan,
- MAX(CASE WHEN month = 'Feb' THEN spend END) AS Feb
- FROM user_spend
- GROUP BY user_id;

6. INNER JOIN vs EXISTS

- SELECT name
- FROM customers c
- WHERE EXISTS (
- SELECT 1 FROM orders o WHERE o.customer_id = c.id
-);

- ✅ `EXISTS` stops at the first match. Good for correlated subqueries.

7. Products never sold

- SELECT p.*
- FROM products p
- LEFT JOIN order_items o ON p.id = o.product_id
- WHERE o.product_id IS NULL;

- ✅ `LEFT JOIN` + NULL check = "no match" rows.

8. Highest order value per customer

- SELECT o.*
- FROM orders o
- WHERE amount = (
- SELECT MAX(amount) FROM orders WHERE customer_id = o.customer_id
-);

- ✅ Correlated subquery per customer.

9. Compare today vs yesterday active users

- WITH daily_users AS (
- SELECT activity_date, COUNT(DISTINCT user_id) AS user_count
- FROM activity_log
- WHERE activity_date IN (CURRENT_DATE, CURRENT_DATE - 1)
- GROUP BY activity_date
-)
- SELECT

10. Recursive CTE for hierarchy

- WITH RECURSIVE org_chart AS (
- SELECT id, parent_id, 1 AS level
- FROM employees
- WHERE parent_id IS NULL
- UNION ALL
- SELECT e.id, e.parent_id, oc.level + 1
- FROM employees e
- JOIN org_chart oc ON e.parent_id = oc.id
-)

11. Rank products by sales within each category

- SELECT product_id, category,
- RANK() OVER (PARTITION BY category ORDER BY total_sales DESC) AS rnk
- FROM product_sales;

- ✅ `RANK()` gives position per category.

12. 7-day rolling avg website visits

- SELECT visit_date,
- AVG(visits) OVER (ORDER BY visit_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS rolling_avg
- FROM web_traffic;

- ✅ Smooth trends using sliding windows.

13. First and last transaction per user

- SELECT *
- FROM (
- SELECT *,
- ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY tx_date ASC) AS rn_asc,
- ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY tx_date DESC) AS rn_desc
- FROM transactions
-) t

14. Detect gaps in sequential dates

- SELECT date,
- LAG(date) OVER (ORDER BY date) AS prev_date,
- date - LAG(date) OVER (ORDER BY date) AS gap
- FROM events;

- ✅ Easily identify missing dates or IDs.

15. Cumulative spend per customer per month

- SELECT customer_id, month,
- SUM(spend) OVER (PARTITION BY customer_id ORDER BY month) AS cum_spend
- FROM monthly_spend;

- ✅ Running totals within groups.