Clase 044 — SQL avanzado: CTEs, window functions, subqueries correlacionadas

Parte: 0 — Prerrequisitos · Fuente: Tanimura, SQL for Data Scientists caps. 4-5 · PostgreSQL docs (window functions). · Duración estimada: 120 min.

Clase 044 — SQL avanzado: CTEs, window functions, subqueries correlacionadas

> *Parte: 0 — Prerrequisitos · Fuente: Tanimura, SQL for Data Scientists caps. 4-5 · PostgreSQL docs (window functions). · Duración estimada: 120 min.*

Objetivo

Que el alumno escriba SQL legible y potente: CTEs (WITH) para descomponer queries complejas, window functions (OVER) para rankings/totales corridos/lag/lead sin perder filas, y subqueries correlacionadas cuando aportan.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Escribir CTEs con WITH name AS (...) para mejorar legibilidad.
2. Encadenar múltiples CTEs: WITH a AS (...), b AS (...) SELECT ....
3. Aplicar window functions: ROW_NUMBER(), RANK(), LAG(), LEAD(), SUM() OVER (PARTITION BY ... ORDER BY ...).
4. Calcular ranking por grupo con ROW_NUMBER() OVER (PARTITION BY ...).
5. Diferenciar subquery (independiente) vs correlacionada (depende de la outer).

Temas

#	Tema	Por qué importa
1	CTEs: WITH name AS (...)	Descomponer queries largas.
2	Múltiples CTEs encadenadas	Pipeline legible.
3	Recursive CTEs	Jerarquías, grafos.
4	Window functions: OVER (PARTITION BY ...	Agregar sin colapsar filas.
5	ROW_NUMBER, RANK, DENSE_RANK	Diferencias sutiles.
6	LAG, LEAD: comparar con fila anterior/sigu	Series temporales.
7	Subqueries correlacionadas	Cuando la subquery depende de la outer.

Definiciones y características

CTE (Common Table Expression)

: Vista temporal dentro de una query con WITH nombre AS (...). Descompone queries complejas en pasos legibles. Puedes encadenar múltiples: WITH a AS (...), b AS (...) SELECT ....

Recursive CTE

: CTE que se referencia a sí misma. Útil para jerarquías (árbol organizacional), grafos, generar series (calendario diario). Sintaxis: WITH RECURSIVE t AS (caso_base UNION ALL caso_recursivo).

Window function

: Agregación que no colapsa filas — añade el resultado por fila. Sintaxis: FUNC() OVER (PARTITION BY col ORDER BY col2). Ejemplos: ROW_NUMBER, RANK, LAG, LEAD, SUM() OVER (...).

PARTITION BY vs GROUP BY

: PARTITION BY (en window): subgrupos para la función, pero mantiene cada fila. GROUP BY: reduce a una fila por grupo.

LAG / LEAD

: Acceden a la fila anterior/siguiente dentro de la partition. LAG(monto, 1) OVER (PARTITION BY cliente ORDER BY fecha). Útil para diffs, growth rates.

Subquery correlacionada

: Subquery que depende de la outer query (referencia sus columnas). Se ejecuta una vez por cada fila de la outer. Más lenta que JOIN equivalente.

Dataset / recursos

SQLite con ordenes (cliente_id, fecha, monto) de clase 041 — extendido. Sin descarga.

Ejercicios

1. CTE básica. Reescribe una query con subquery anidada usando WITH.

2. ROW_NUMBER por grupo. Top-1 orden por cliente (mayor monto).

3. Total corrido. SUM(monto) OVER (PARTITION BY cliente_id ORDER BY fecha) — total acumulado por cliente.

4. LAG. Por cliente, diferencia entre el monto actual y el anterior.

5. Recursive CTE. Genera serie de fechas día a día desde 2024-01-01 a 2024-01-31.

Homework verificable

Notebook: (a) 3 versiones de la misma query (anidada → CTE → CTEs múltiples) comparando legibilidad; (b) top-3 órdenes por cliente con ROW_NUMBER; (c) total corrido y delta vs orden anterior; (d) recursive CTE para calendario diario.

Criterio de aceptación: Las 3 versiones devuelven exactamente el mismo resultado. Window functions sin error.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
syntax error at or near "OVER"	Motor sin soporte de window functions (SQL
ROW_NUMBER() da números repetidos	Olvidaste OVER (...). Sin él, no es window
CTE recursiva nunca termina	Caso base falta o caso recursivo no conver
LAG(x) OVER (ORDER BY fecha) da NULL en la	Comportamiento esperado — no hay fila ante
CTE da mismo resultado pero más lento que	Algunos motores no inlineaban CTEs (Postgr

Preguntas frecuentes

¿CTE o subquery?

CTE si el lector necesita entender qué hace cada paso (legibilidad). Subquery si es trivial y de un solo uso. Para queries >10 líneas, CTE casi siempre gana.

¿ROW_NUMBER, RANK o DENSE_RANK?

Para valores [10, 20, 20, 30]: ROW_NUMBER [1,2,3,4] (siempre único). RANK [1,2,2,4] (huecos). DENSE_RANK [1,2,2,3] (sin huecos). Elige según semántica.

¿Window function es lo mismo que groupby+merge en pandas?

Conceptualmente sí — g.transform(...) en pandas hace lo equivalente. Window functions son la versión SQL más eficiente.

¿Cuándo subquery correlacionada vs JOIN?

Casi siempre JOIN o window function es más rápido. Correlacionada solo cuando no tiene equivalente JOIN (raro) o el optimizador del motor la maneja bien (motores modernos).

¿Top-N por grupo?

Patrón estándar: WITH ranked AS (SELECT , ROW_NUMBER() OVER (PARTITION BY grupo ORDER BY metric DESC) rn FROM tabla) SELECT FROM ranked WHERE rn <= N.

Referencias

- Tanimura, SQL for Data Scientists, caps. 4-5.
- PostgreSQL window functions tutorial
- Modern SQL — CTEs

Siguiente clase

Clase 045 — SQL desde Python: sqlite3, SQLAlchemy, DuckDB

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import sqlite3
import pandas as pd

con = sqlite3.connect(':memory:')
con.executescript('''
CREATE TABLE ordenes (
    orden_id INTEGER PRIMARY KEY,
    cliente_id INTEGER,
    fecha DATE,
    monto REAL
);
INSERT INTO ordenes (cliente_id, fecha, monto) VALUES
    (1, '2024-01-15', 120),(1, '2024-02-20',  80),(1, '2024-03-12', 150),(1, '2024-04-05',  60),
    (2, '2024-02-10',  40),(2, '2024-03-15',  70),(2, '2024-04-22',  50),
```

```
    (3, '2024-01-22', 200),(3, '2024-03-15', 180),(3, '2024-04-01', 220),
    (5, '2024-02-28', 300),(5, '2024-03-22',  90),(5, '2024-04-15', 400);
""")
```

Archivos complementarios

- notebook.ipynb