Día 4: Seguridad en PostgreSQL + SQL Avanzado (Subqueries & CTE)

Checklist	☑
# Día	4
Estado	Completado
Fecha	@22 de diciembre de 2025
Notas Técnicas	**1. Roles vs Usuarios en PostgreSQL:** En PostgreSQL no existe una entidad separada llamada “usuario”. Todo es un **rol**. Un rol se considera usuario cuando tiene el atributo `LOGIN`. Esto permite manejar permisos y autenticación de forma unificada y flexible a nivel de cluster. **2. Alias Obligatorio en Subqueries:** Al utilizar una subconsulta en la cláusula `FROM`, es **obligatorio** asignar un alias (por ejemplo, `SELECT ... FROM (...) AS mi_subquery`). Si se omite, PostgreSQL arrojará un error de sintaxis, ya que necesita un nombre para referenciar esa "tabla temporal". **3. Diferencia práctica entre Subqueries y CTE:** Las subqueries son útiles para lógica puntual, pero cuando una consulta crece en complejidad, los **CTE (WITH)** permiten dividir el problema en pasos lógicos, mejorando la legibilidad, el mantenimiento y la colaboración en equipos profesionales. **4. CTE permite:** Los CTE permiten estructurar consultas complejas de forma

	clara y reutilizable. Mejoran la legibilidad del SQL, facilitan el mantenimiento y son preferidos en entornos profesionales frente a subqueries anidadas.
Tiempo Invertido	5 horas

Seguridad en PostgreSQL + SQL Avanzado (Subqueries & CTE)

🎯 Objetivo del día

1. Completar correctamente la **gestión de permisos y validación de roles** (pendiente del Día 3).
2. Aprender y practicar **Subqueries** y **CTE (WITH)** en PostgreSQL.
3. Documentar resultados de forma clara y profesional.

🧱 BLOQUE 1

Cierre del Día 3: Permisos y Validación (Prioridad ALTA)

Para saber cuales son los Roles y Usuarios (entiendase usuario, como un rol que puede hacer Login) ya existentes, se puede hacer la siguiente consulta SQL:

```
--Concultar los nombres de los ROLES que hay
SELECT
rolname,
rolsuper,
rolcreatedb,
rolcreaterole,
rolcanlogin,
rolinherit
```

```
FROM pg_roles
ORDER BY rolname;
```

Lo cual arrojará una tabla con las información solicitada

	rolname name	rolsuper boolean	rolcreatedb boolean	rolcreaterole boolean	rolcanlogin boolean	rolinherit boolean
1	admin	true	false	false	true	true
2	administrador_test	false	true	false	true	true
3	escritura	false	false	false	false	true
4	lector	false	false	false	false	true
5	lectura	false	false	false	false	true
6	pg_checkpoint	false	false	false	false	true
7	pg_create_subscription	false	false	false	false	true
8	pg_database_owner	false	false	false	false	true
9	pg_execute_server_program	false	false	false	false	true
10	pg_monitor	false	false	false	false	true
11	pg_read_all_data	false	false	false	false	true
12	pg_read_all_settings	false	false	false	false	true
13	pg_read_all_stats	false	false	false	false	true
14	pg_read_server_files	false	false	false	false	true
15	pg_signal_backend	false	false	false	false	true
16	pg_stat_scan_tables	false	false	false	false	true
17	pg_use_reserved_connectio...	false	false	false	false	true
18	pg_write_all_data	false	false	false	false	true
19	pg_write_server_files	false	false	false	false	true
20	postgres	true	true	true	true	true
21	usuario_escritura	false	false	false	true	true

Para **consultar** cuales son los **Usuarios**, se debe poner la condición de **WHERE** donde **Loging** sea **TRUE**, esta sintaxis debe ir **antes** de especificar **ORDER BY**

```
--Consultar roles que pueden hacer Login,o sea los Usuarios
SELECT
rolname,
```

```
rolsuper,
rolcreatedb,
rolcreaterole,
rolcanlogin,
rolinherit
FROM pg_roles
WHERE rolcanlogin = true
ORDER BY rolname;
```

El resultado será una tabla con todos los Roles que pueden hacer Login

	rolname name	rolsuper boolean	rolcreatedb boolean	rolcreaterole boolean	rolcanlogin boolean	rolinherit boolean
1	admin	true	false	false	true	true
2	administrador_test	false	true	false	true	true
3	postgres	true	true	true	true	true
4	usuario_escritura	false	false	false	true	true
5	usuario_escritura2	false	true	true	true	true
6	usuario_lectura	false	false	false	true	true

Esta información también puede ser adquirida desde la terminal psql, usando el comando `\du`

```
postgres=# \du
                                   Lista de roles
   Nombre de rol    |                         Atributos
--------------------+------------------------------------------------------------
 admin              | Superusuario                                               +
                    | Constrase±a vßlida hasta 2025-12-31 19:30:59-05
 administrador_test | Crear BD                                                   +
                    | Constrase±a vßlida hasta 2025-12-31 13:00:00-05
 escritura          | No puede conectarse
 lector             | No puede conectarse
 lectura            | No puede conectarse
 postgres           | Superusuario, Crear rol, Crear BD, Replicaci%n, Ignora RLS
 usuario_escritura  |
 usuario_escritura2 | Crear rol, Crear BD
 usuario_lectura    |
```

🗂️ Permisos sobre schema:

- Conceder USAGE sobre schema `app` al rol `lector`
- Conceder USAGE sobre schema `app` al rol `escritura`

```
GRANT USAGEON SCHEMA app TO lector;
GRANT USAGEON SCHEMA app TO escritura;
```

```sql
--Permiso sobre Schema:

GRANT USAGE ON SCHEMA app TO lector;
GRANT USAGE ON SCHEMA app TO escritura;
```

📖 Permisos sobre tablas:

- Conceder SELECT al rol `lector`

```
GRANT SELECT ON app.datos TO lector;
```

```sql
33  --Permiso sobre Tablas:
34
35  GRANT SELECT ON app.datos TO lector;
```

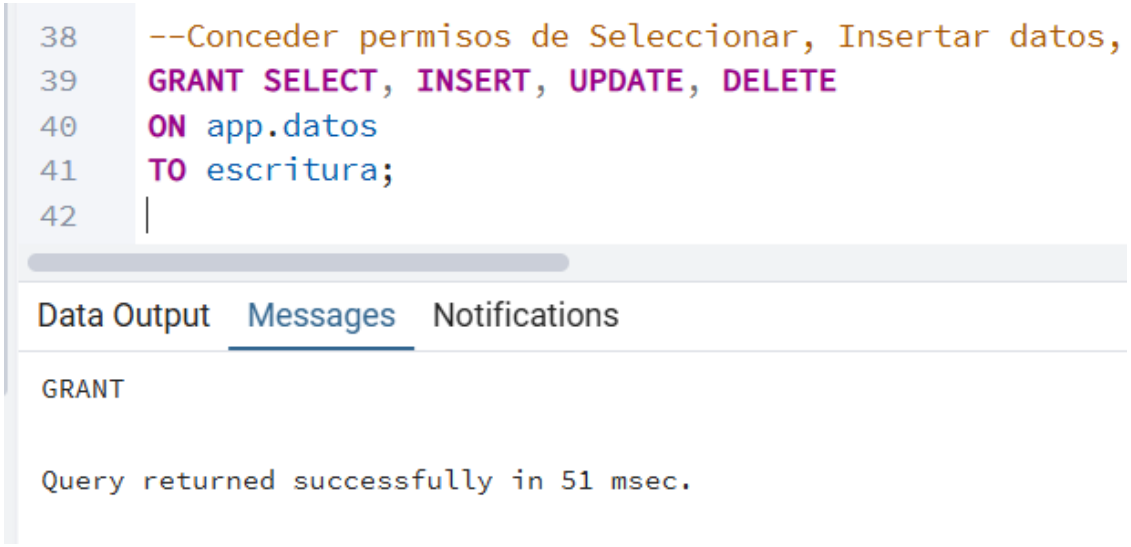
Data Output | Messages | Notifications

```
GRANT

Query returned successfully in 67 msec.
```

- Conceder SELECT, INSERT, UPDATE, DELETE al rol `escritura`

```
GRANT SELECT,INSERT,UPDATE,DELETE
ON app.datos
TO escritura;
```



En el cuadro se puede ver el desglose de cada parte de la sentencia SQL

Parte	Significado
`GRANT`	Conceder
`SELECT`	Permiso
`ON app.datos`	Sobre qué objeto
`TO lector`	A qué rol

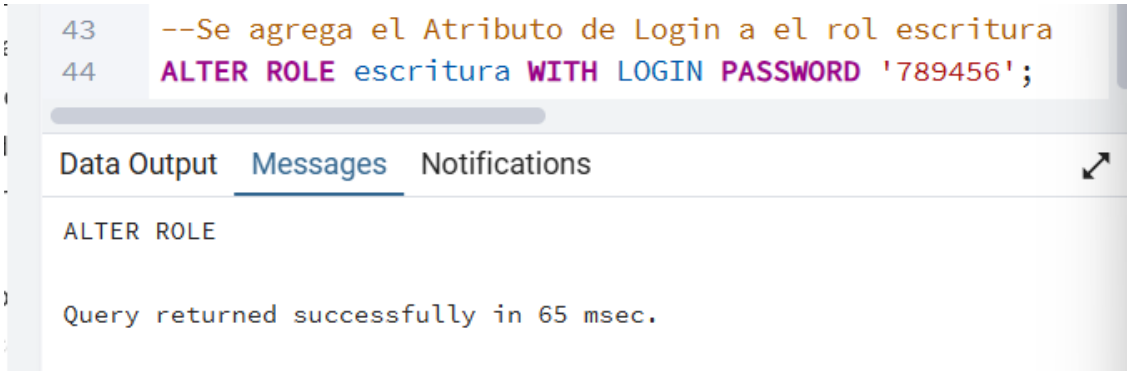
🔍 **Validación:**

Antes de hacer la validación de los permisos, agregué el **Atributo de Login** a los R**oles de lector y escritura**, ahora son **Usuarios**.

- Desde la terminal psql:

```
postgres=# ALTER ROLE lector WITH LOGIN PASSWORD '789456';
ALTER ROLE
postgres=#
```

- Desde pgAdmin:



1. **Usuario lector**

- Probar SELECT (**debe funcionar**)

```
SELECT*FROM app.datos;
```

```
alquiler=> SELECT * FROM app.datos;
           tipo            |           colonia           | habitaciones | a
rea  |   valor
---------------------------+-----------------------------+--------------+--
-----+------------
 Cocineta                  | Condesa                     |            1 |
  40 |     5950.0
 Casa                      | Polanco                     |            2 |
 100 |    24500.0
 Conjunto Comercial/Sala   | Santa Fe                    |            0 |
 150 |    18200.0
 Departamento              | Centro Hist%rico            |            1 |
  15 |     2800.0
 Departamento              | Del Valle                   |            1 |
  48 |     2800.0
 Casa de Condominio        | Santa Fe                    |            5 |
 750 |    77000.0
```

- Probar INSERT (**debe fallar**)

```
INSERT INTO app.datos (...)VALUES (...);
```

```
alquiler=> INSERT INTO app.datos (tipo, colonia, habitaciones, area, valor)
VALUES ('Casa de Campo', 'Roma', 5, 50,10000);
ERROR:  permiso denegado a la tabla datos
alquiler=>
```

2. Usuario escritura

- Probar INSERT (**debe funcionar**)

```
INSERT INTO app.datos (tipo, colonia, habitaciones, area, valor)
VALUES ('Apartamento','Centro',2,60,1200);
```

```
alquiler=> INSERT INTO app.datos (tipo, colonia, habitaciones, area, valor)
VALUES ('Casa de Campo', 'Roma', 5, 50,10000);
INSERT 0 1
alquiler=>
```

🧱 BLOQUE 2

Subqueries

Una **subquery** (o subconsulta) en PostgreSQL es una consulta SQL que está anidada dentro de otra consulta más grande, las subqueries son extremadamente flexibles y pueden aparecer en casi cualquier parte de una sentencia: `SELECT`, `INSERT`, `UPDATE` o `DELETE`.

- **En el `WHERE`:** Para filtrar datos basados en el resultado de otra búsqueda.
- **En el `FROM`:** Tratando a la subconsulta como si fuera una tabla temporal (también llamada *derived table*).
- **En el `SELECT`:** Para generar una columna calculada sobre la marcha.
- **En el `HAVING`:** Para filtrar grupos de datos.

Tipos principales de Subqueries

Tipo	Descripción	Resultado
Escalar	Devuelve un único valor (una fila, una columna).	Un dato (ej. un ID o un precio).
De fila	Devuelve una sola fila con múltiples columnas.	Una estructura `(valor1, valor2)`.
De tabla	Devuelve una o más columnas con múltiples filas.	Se usa con operadores como `IN`, `ANY` o `ALL`.

Diferencia entre: Correlacionadas vs. No Correlacionadas

1. **Subquery No Correlacionada:** Es independiente. Se ejecuta una sola vez, obtiene el resultado y se lo entrega a la consulta principal. Es rápida y eficiente.
   - *Ejemplo:* Buscar empleados que ganen más que el promedio (el promedio se calcula una sola vez).
2. **Subquery Correlacionada:** La subconsulta hace referencia a una columna de la consulta principal. Esto obliga a PostgreSQL a ejecutar la subconsulta **una vez por cada fila** procesada por la consulta principal.

- *Ejemplo:* Buscar empleados que ganen más que el promedio *de su propio departamento*.

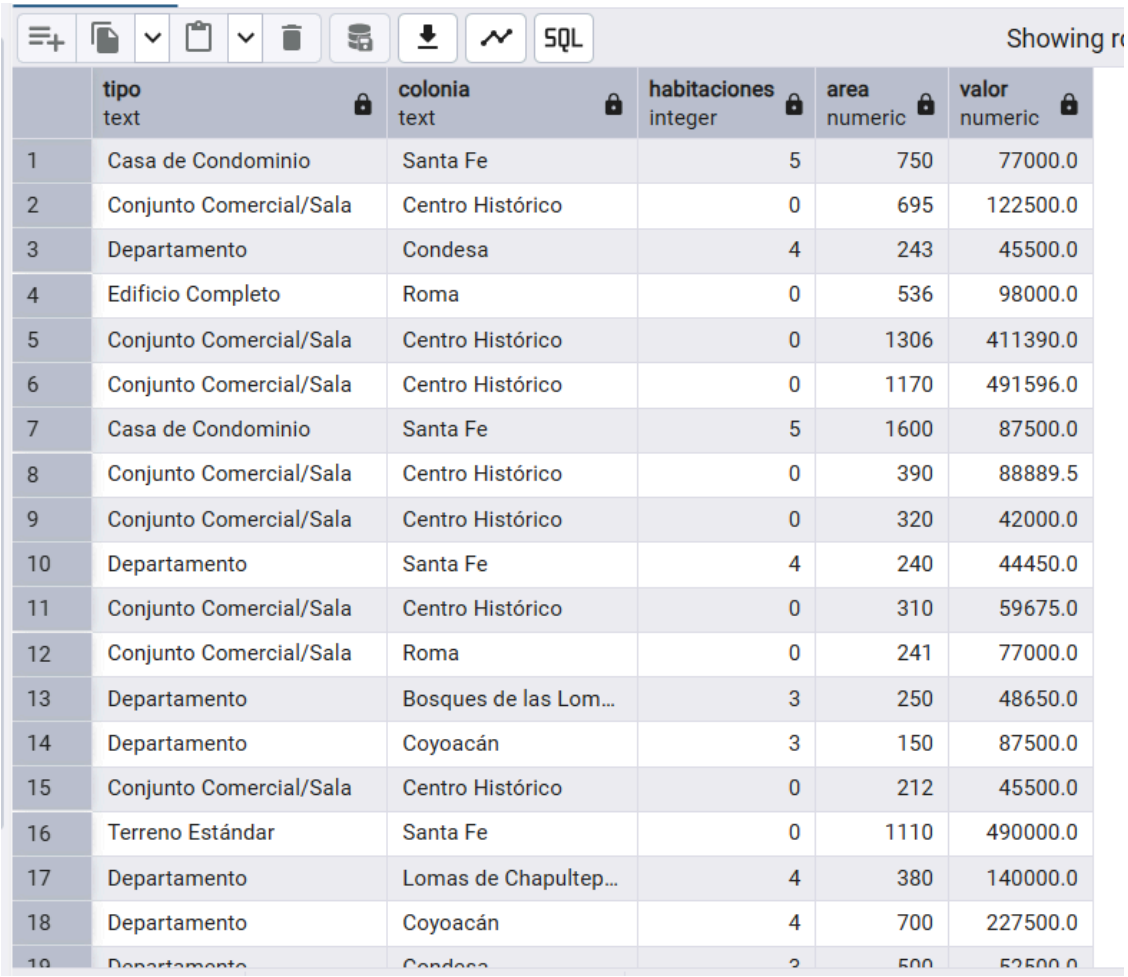
✍️ **Ejercicios prácticos:**

1. **Subquery en WHERE**

```
--Trae propiedades con valor mayor al promedio.
SELECT*
FROM app.datos
WHERE valor> (
SELECT AVG(valor)
FROM app.datos
);
```

```sql
--Ejercicios de Subquery

--Subquery en WHERE
--Trae propiedades con valor mayor al promedio.
SELECT * FROM app.datos
WHERE valor > (
SELECT AVG(valor)
FROM app.datos
);
```

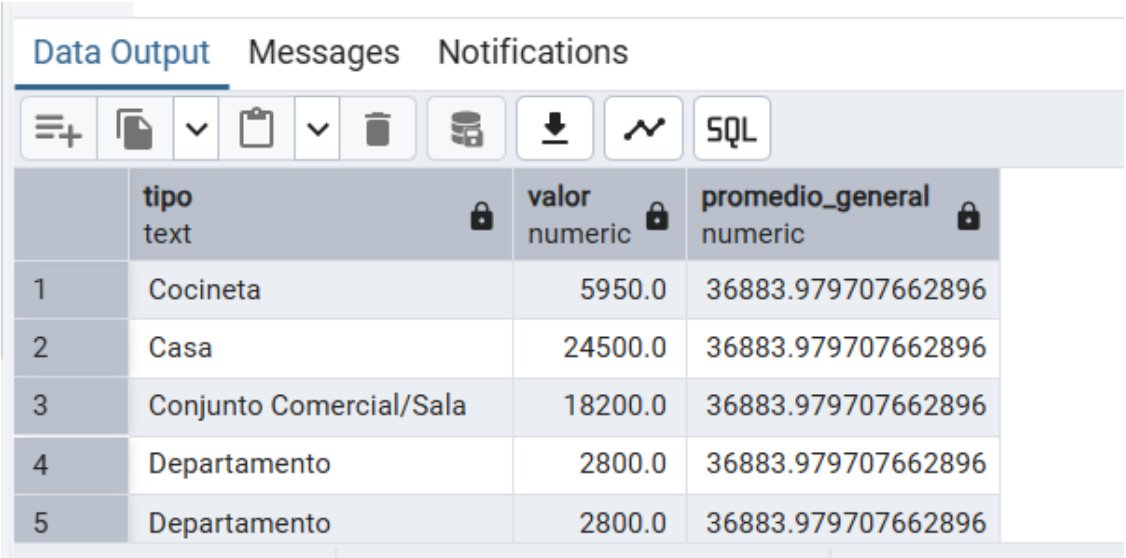


	tipo text	colonia text	habitaciones integer	area numeric	valor numeric
1	Casa de Condominio	Santa Fe	5	750	77000.0
2	Conjunto Comercial/Sala	Centro Histórico	0	695	122500.0
3	Departamento	Condesa	4	243	45500.0
4	Edificio Completo	Roma	0	536	98000.0
5	Conjunto Comercial/Sala	Centro Histórico	0	1306	411390.0
6	Conjunto Comercial/Sala	Centro Histórico	0	1170	491596.0
7	Casa de Condominio	Santa Fe	5	1600	87500.0
8	Conjunto Comercial/Sala	Centro Histórico	0	390	88889.5
9	Conjunto Comercial/Sala	Centro Histórico	0	320	42000.0
10	Departamento	Santa Fe	4	240	44450.0
11	Conjunto Comercial/Sala	Centro Histórico	0	310	59675.0
12	Conjunto Comercial/Sala	Roma	0	241	77000.0
13	Departamento	Bosques de las Lom...	3	250	48650.0
14	Departamento	Coyoacán	3	150	87500.0
15	Conjunto Comercial/Sala	Centro Histórico	0	212	45500.0
16	Terreno Estándar	Santa Fe	0	1110	490000.0
17	Departamento	Lomas de Chapultep...	4	380	140000.0
18	Departamento	Coyoacán	4	700	227500.0
19	Departamento	Condesa	3	500	52500.0

2. Subquery en SELECT

```
SELECT
  tipo,
  valor,
  (SELECT AVG(valor)FROM app.datos)AS promedio_general
FROM app.datos;
```

```sql
--Subquery en SELECT
--Genera una vista con las columnas Tipo, Valor y crea una con el Promedio General
SELECT tipo, valor,
(SELECT AVG(VALOR) FROM app.datos) AS promedio_general
FROM app.datos;
```



Data Output | Messages | Notifications

SQL

	tipo text	valor numeric	promedio_general numeric
1	Cocineta	5950.0	36883.979707662896
2	Casa	24500.0	36883.979707662896
3	Conjunto Comercial/Sala	18200.0	36883.979707662896
4	Departamento	2800.0	36883.979707662896
5	Departamento	2800.0	36883.979707662896

3. Subquery en FROM

```
SELECT tipo,AVG(valor)
FROM (
SELECT tipo, valor
FROM app.datos
) sub
GROUP BY tipo;
```

Desgloce de la cosulta:

`(SELECT tipo, valor FROM app.datos) sub`

- **¿Qué hace?**: Primero, PostgreSQL ejecuta lo que está entre paréntesis. Va a la tabla `datos` (que está dentro del esquema llamado `app`) y extrae todas las filas, pero solo las columnas `tipo` y `valor`.
- **El alias** `sub`: Es obligatorio ponerle un nombre (en este caso se eligió "sub") a cualquier subquery que pongas en el `FROM`. Es como decirle a Postgres: "Trata este resultado temporal como si fuera una tabla llamada *sub*".

`SELECT tipo, AVG(valor) ... GROUP BY tipo;`

- `AVG(valor)`: Toma los datos que vienen de la subquery y calcula el promedio de la columna `valor`.

- `GROUP BY tipo`: Esta es la clave. Le dice a la base de datos: "No me des un solo promedio global; agrúpame los resultados por cada categoría que encuentres en la columna `tipo`".

Ejemplo visual de lo que hace:

Si la tabla `app.datos` fuera así:

tipo	valor
A	10
B	20
A	20
B	40

El resultado final de tu consulta sería:

tipo	avg
A	15
B	30

```sql
--Subquery en From
--calcula el promedio de Valor para cada Tipo que existe en la tabla app.datos
SELECT tipo, AVG(valor)
FROM (
SELECT tipo, valor
FROM app.datos
)sub
GROUP BY tipo;
```

Data Output | Messages | Notifications

Showing rows: 1 to 22 | Page No: 1

	tipo text	avg numeric
1	Casa de Condominio	51428.609836065574
2	Conjunto Comercial/Sala	57675.214549938348
3	Terreno Estándar	135364.444444444444
4	Tienda/Salón	56263.544251824818
5	Loft	9924.5263157894736842

🧱 BLOQUE 3

SQL Avanzado: CTE (WITH)

Un **CTE (Common Table Expression)** es una consulta temporal con nombre, definida con la cláusula `WITH`, que existe solo durante la ejecución de la query, funciona como una tabla virtual reutilizable dentro de una consulta.

La sintaxis básica es:

```
WITH cte_name AS (
   SELECT ...
)
SELECT *
FROM cte_name;
```

```
WITH mi_tabla_temporal AS (
   SELECT columna1, columna2
   FROM tabla_real
   WHERE condicion = true
)
SELECT * FROM mi_tabla_temporal; -- Aquí usas el CTE
```

La **Subquery** es como un "**paréntesis**" en medio de una oración, y un **CTE** es como definir una **variable** al principio del texto para usarla más adelante. Se definen usando la palabra clave `WITH`

Ventajas de usar CTE:

Característica	Subquery	CTE (`WITH`)
Lectura	"De adentro hacia afuera" (difícil de seguir).	"De arriba hacia abajo" (lineal y lógica).
Reutilización	Tienes que copiar y pegar el código si lo usas dos veces.	Lo defines una vez y lo invocas cuantas veces quieras.
Recursividad	No permite recursión.	Permite `WITH RECURSIVE` (ideal para organigramas o rutas).
Mantenimiento	Muy difícil de depurar si hay muchos niveles.	Muy fácil de aislar y probar por partes.

Nota: Los CTE permiten estructurar consultas complejas de forma clara y reutilizable. Mejoran la legibilidad del SQL, facilitan el mantenimiento y son preferidos en entornos profesionales frente a subqueries anidadas.

Checklist:

- ☑️ ~~Estudio del tema~~
- ☑️ ~~Ejecución de scripts~~
- ☑️ ~~Evidencia guardada~~
- ☑️ ~~Notas técnicas escritas~~

📺 **Recursos:**

- https://youtu.be/xuASGBwNboU?si=zyDXYBXvKo8y256u
- https://youtu.be/79pM5FwuW4U?si=MBpsdKq_kxCiCPo4