Día 6: SQL profesional (Enfoque 100% Práctico)

Checklist	☑
# Día	6
Estado	En progreso
Fecha	@30 de diciembre de 2025
Notas Técnicas	**1. Encapsulamiento de Window Functions mediante Subconsultas:** Es necesario utilizar una subconsulta en la cláusula `FROM` (o una CTE) para encapsular el cálculo de funciones como `ROW_NUMBER()`. Esto se debe a que el flujo lógico de ejecución de SQL procesa el `WHERE` antes que el `SELECT`, impidiendo filtrar directamente por un alias de ranking generado en el mismo nivel. **2. Unicidad Secuencial con ROW_NUMBER():** A diferencia de otras funciones de clasificación, `ROW_NUMBER()` garantiza una numeración secuencial única (1, 2, 3...) para cada fila dentro de su partición. Esta función ignora si existen empates en los valores de ordenamiento, lo que la hace ideal para seleccionar un número exacto de registros (ej. un "Top 3") por categoría. **3. Optimización de Consultas mediante Índices:** El uso de `CREATE INDEX` permite generar estructuras auxiliares que aceleran drásticamente las operaciones de búsqueda, filtrado y ordenamiento. Al aplicar un índice sobre una columna como `colonia`, se evita que el motor tenga que examinar cada fila de la tabla de forma secuencial.

	4. Sequential Scan vs. Index Scan: Un aspecto crítico del rendimiento es el método de acceso a los datos. Un `Sequential Scan` recorre toda la tabla, lo cual es ineficiente en conjuntos de datos grandes, mientras que un `Index Scan` localiza los datos directamente a través del índice, mejorando significativamente el tiempo de respuesta. **5. Diagnóstico de Rendimiento con EXPLAIN ANALYZE:** Esta herramienta es indispensable para el análisis técnico, ya que no solo muestra el plan de ejecución elegido por PostgreSQL, sino que ejecuta la consulta para reportar el costo real y el tiempo invertido. Permite validar si el motor está utilizando efectivamente los índices creados.
⩵ Tiempo Invertido	5 horas

📆 Día 6 – SQL Profesional (Enfoque 100% Práctico)

🎯 Objetivo del día

Afianzar los conceptos de SQL avanzado vistos hasta ahora mediante **casos reales**, utilizando **tipos de datos avanzados**, buenas prácticas y consultas con intención profesional.

🔷 BLOQUE 1 – Preparación del entorno:

✅ Tareas:

1. Confirmar conexión a PostgreSQL

Se logró la conexión a la Base de Datos Alquiler por medio de la Terminal de SQL Shell (psql).

```
Server [localhost]:
Database [postgres]: alquiler
Port [5432]:
Username [postgres]:
Contraseña para usuario postgres:

psql (16.11)
ADVERTENCIA: El código de página de la consola (850) difiere del código
             de página de Windows (1252).
             Los caracteres de 8 bits pueden funcionar incorrectamente.
             Vea la página de referencia de psql «Notes for Windows users»
             para obtener más detalles.
Digite «help» para obtener ayuda.

alquiler=#
```

2. Usar schema `app`
3. Verificar tabla `app.datos`

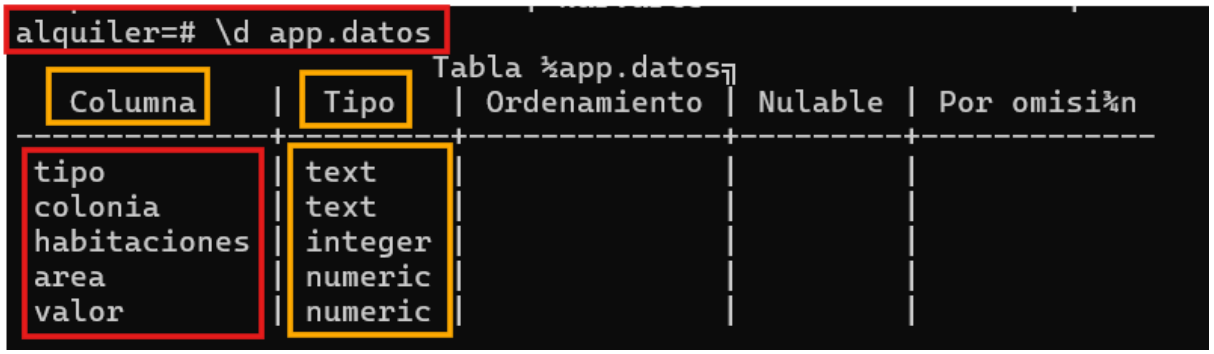
Se accede a la información de la Tabla Datos que se encuentra en el Schema App de la Base de Datos Alquiler, esta tabla consta de 5 columnas que son tipo, colonia, habitaciones, area y valor.

```
alquiler=# SELECT * FROM app.datos;
           tipo            |          colonia          | habitaciones | area  |   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
 Conjunto Comercial/Sala   | Centro Hist%rico          |            0 |   695 |   122500.0
 Departamento              | Centro Hist%rico          |            1 |    36 |     4200.0
 Departamento              | Condesa                   |            1 |    40 |     7000.0
 Cocineta                  | Condesa                   |            1 |    27 |     6300.0
 Departamento              | Condesa                   |            4 |   243 |    45500.0
 Edificio Completo         | Roma                      |            0 |   536 |    98000.0
 Departamento en Hotel     | Roma                      |            3 |    80 |    13300.0
 Departamento              | Santa Fe                  |            2 |    67 |     5950.0
 Departamento              | Narvarte                  |            2 |   110 |     6650.0
 Casa de Condominio        | Santa Fe                  |            4 |   466 |    26250.0
 Departamento              | Narvarte                  |            2 |    78 |     7000.0
 Departamento en Hotel     | Lomas de Chapultepec      |            1 |    56 |    15750.0
 Departamento              | Coyoacßn                  |            3 |   125 |    24500.0
 Departamento              | Narvarte                  |            1 |    48 |     2625.0
 Departamento              | Santa Fe                  |            3 |   175 |    14700.0
 Departamento              | Roma                      |            2 |    76 |     8750.0
```

4. Revisar tipos de datos actuales

```
\d app.datos;
```

Se observa que las columnas tipo y colonia poseen datos de tipo text, la columna habitaciones es del tipo integer y para las columnas area y valor los datos debes ser numeric.



🔷 BLOQUE 2 – Tipos de datos avanzados:

🧪 Ejercicios:

1. Revisión crítica de tipos:

Identifica columnas que podrían mejorar:

- `tipo` → `VARCHAR`
- `colonia` → `VARCHAR`
- `valor` → `NUMERIC(12,2)`
- `area` → `NUMERIC(8,2)`

```
--Cambiar el tipo de datos de las columnas tipo, colonia, valor y area
--tipo text --> VARCHAR
--colonia text --> VARCHAR
--valor numeric --> NUMERIC (12,2)
--area numeric --> NUMERIC (8,2)

ALTER TABLE app.datos
```

```
ALTER COLUMN tipo TYPE VARCHAR;

ALTER TABLE app.datos
ALTER COLUMN colonia TYPE VARCHAR;

ALTER TABLE app.datos
ALTER COLUMN valor TYPE NUMERIC(12,2);

ALTER TABLE app.datos
ALTER COLUMN area TYPE NUMERIC(8,2);
```

Se hacen los cambios pertinentes del tipo de dato de las columnas tipo, colonia, valor y area.

```sql
--Cambiar el tipo de datos de las columnas tipo, co
--tipo text --> VARCHAR
--colonia text --> VARCHAR
--valor numeric --> NUMERIC (12,2)
--area numeric --> NUMERIC (8,2)

ALTER TABLE app.datos
ALTER COLUMN tipo TYPE VARCHAR;

ALTER TABLE app.datos
ALTER COLUMN colonia TYPE VARCHAR;

ALTER TABLE app.datos
ALTER COLUMN valor TYPE NUMERIC(12,2);

ALTER TABLE app.datos
ALTER COLUMN area TYPE NUMERIC(8,2);
```

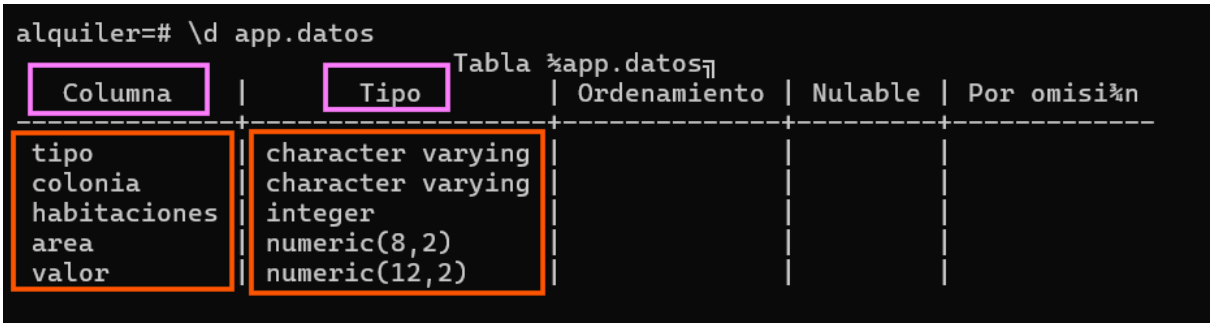
Data Output | Messages | Notifications

```
ALTER TABLE

Query returned successfully in 122 msec.
```

```
alquiler=# \d app.datos
                Tabla %app.datos┐
  Columna    | Tipo    | Ordenamiento | Nulable | Por omisi%n
-------------+---------+--------------+---------+-------------
tipo         | text    |              |         |
colonia      | text    |              |         |
habitaciones | integer |              |         |
area         | numeric |              |         |
valor        | numeric |              |         |
```

Antes de hacer el cambio del tipo de datos de las columnas



Después de hacer el cambio del tipo de datos de las columnas

2. Uso de `COALESCE` :

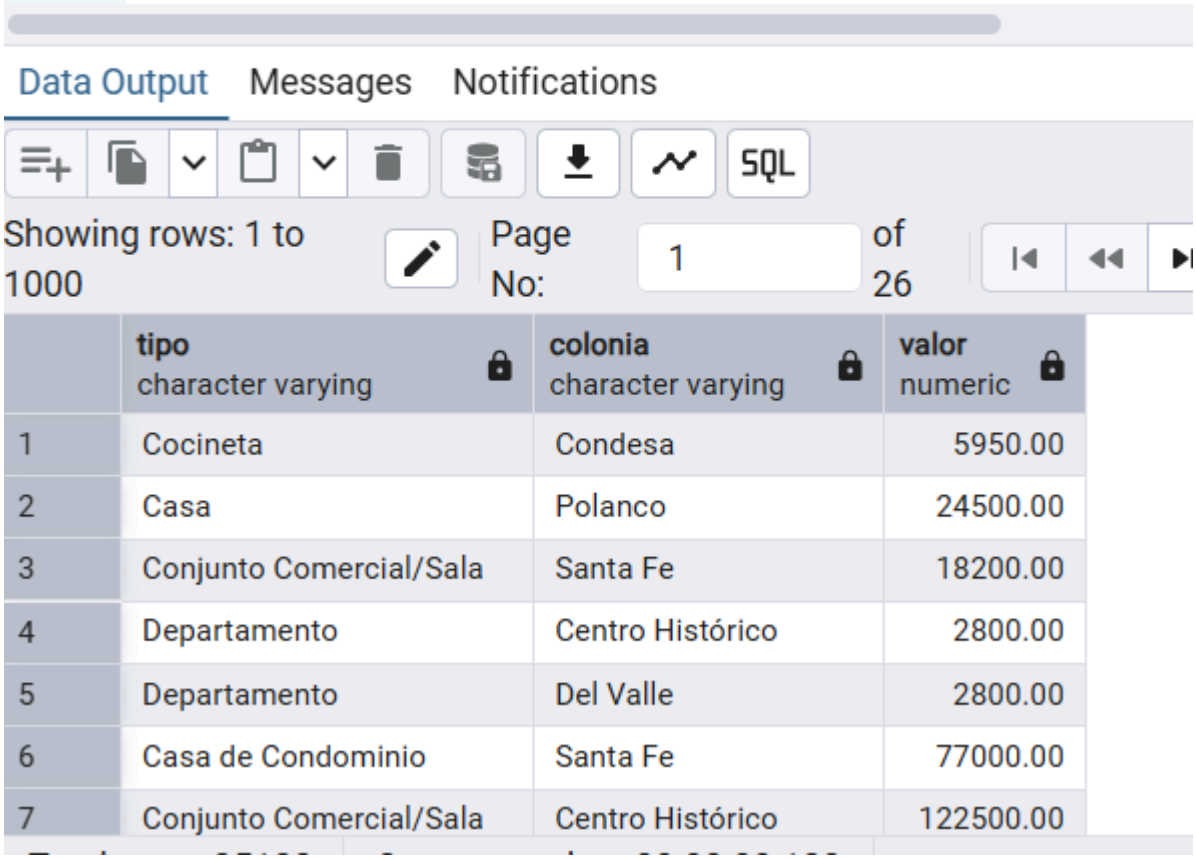
Manejo profesional de `NULL`.

```
SELECT
  tipo,
COALESCE(colonia,'SIN COLONIA')AS colonia,
COALESCE(valor,0)AS valor
FROM app.datos;
```

`COALESCE(colonia,'SIN COLONIA')AS colonia,` : Si colonia tiene un valor en esa fila devuelve ese dato, el nombre de la colonia, pero si es NULL devolverá SIN COLONIA

`COALESCE(valor,0)AS valor` : Si la información de valor es un número se mostrará ese núero, de ser NULL se mostra 0

```sql
SELECT
  tipo,
COALESCE(colonia,'SIN COLONIA')AS colonia,
COALESCE(valor,0)AS valor
FROM app.datos;
```



Data Output | Messages | Notifications

Showing rows: 1 to 1000 | Page No: 1 of 26

	tipo character varying	colonia character varying	valor numeric
1	Cocineta	Condesa	5950.00
2	Casa	Polanco	24500.00
3	Conjunto Comercial/Sala	Santa Fe	18200.00
4	Departamento	Centro Histórico	2800.00
5	Departamento	Del Valle	2800.00
6	Casa de Condominio	Santa Fe	77000.00
7	Conjunto Comercial/Sala	Centro Histórico	122500.00

🔷 **BLOQUE 3 – SQL profesional aplicado:**

Caso 1:

Clasificación de propiedades

```sql
SELECT
  tipo,
  colonia,
  valor,
CASE
WHEN valor>=500000 THEN 'ALTO'
WHEN valor BETWEEN 200000 AND 499999 THEN 'MEDIO'
```

```
ELSE'BAJO'
END AS rango_precio
FROM app.datos;
```

Por medio de la sentencia CASE, se establecen los rangos para la clasificación de las propiedades

```
SELECT
  tipo,
  colonia,
  valor,
CASE
WHEN valor>=500000 THEN 'ALTO'
WHEN valor BETWEEN 200000 AND 499999 THEN 'MEDIO'
ELSE'BAJO'
END AS rango_precio
FROM app.datos;
```

Data Output | Messages | Notifications

	tipo character varying	colonia character varying	valor numeric (12,2)	rango_precio text
577	Conjunto Comercial/Sala	Centro Histórico	41440.00	BAJO
578	Conjunto Comercial/Sala	Santa Fe	2149000.00	ALTO
579	Conjunto Comercial/Sala	Santa Fe	5075.00	BAJO
580	Departamento	Culhuacán CTM	3465.00	BAJO
581	Casa Comercial	Polanco	21000.00	BAJO
582	Departamento	San Pedro de los Pin...	10150.00	BAJO
583	Departamento	Peralvillo	3500.00	BAJO
584	Departamento	Tlalpan	56000.00	BAJO
585	Galpón/Depósito/Almacén	Santa María la Ribera	52500.00	BAJO
586	Tienda/Salón	Villa de Guadalupe	6300.00	BAJO

si el valor de la propiedad es superior a 500.000 es Alto

si el valor está entre 200.000 a 499.999 es Medio

si el valor es inferior a 200.000 es Bajo

Caso 2:

Propiedades por encima del promedio

```
WITH base AS (
SELECT
*,
ROUND(
    AVG(valor) OVER ()
, 2)AS promedio_general
FROM app.datos
)
SELECT*
FROM base
WHERE valor > promedio_general;
```

Se usa CTE + Window Function para traer las propiedades que están por encima del valor promedio.

En el resultado de la consulta, solo está trayendo las propiedades que poseen un valor mayor a 36.883,98 que es el valor promedio de una propiedad en general, sin hacer una segmentación por Colina, cantidad de habitaciones o por el tipo de propiedad.

```sql
WITH base AS (
SELECT
*,
ROUND(
    AVG(valor) OVER ()
, 2)AS promedio_general
FROM app.datos
)
SELECT*
FROM base
WHERE valor > promedio_general;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1000 | Page N

	tipo character varying	colonia character varying	habitaciones integer	area numeric (8,2)	valor numeric (12,2)	promedio_general numeric
1	Casa de Condominio	Santa Fe	5	750.00	77000.00	36883.98
2	Conjunto Comercial/Sala	Centro Histórico	0	695.00	122500.00	36883.98
3	Departamento	Condesa	4	243.00	45500.00	36883.98
4	Edificio Completo	Roma	0	536.00	98000.00	36883.98
5	Conjunto Comercial/Sala	Centro Histórico	0	1306.00	411390.00	36883.98
6	Conjunto Comercial/Sala	Centro Histórico	0	1170.00	491596.00	36883.98
7	Casa de Condominio	Santa Fe	5	1600.00	87500.00	36883.98
8	Conjunto Comercial/Sala	Centro Histórico	0	390.00	88889.50	36883.98

Caso 3:

Top 3 propiedades por colonia

```sql
SELECT*
FROM (
SELECT
   colonia,
   tipo,
   valor,
ROW_NUMBER()OVER (
PARTITION BY colonia
ORDER BY valor DESC
   )AS ranking
FROM app.datos
) t
WHERE ranking<=3;
```

Esta consulta utiliza una **subconsulta en el FROM** para encapsular el cálculo de

`ROW_NUMBER()`. Se prefiere esta estructura (o una CTE) porque el **flujo lógico de SQL** procesa el `WHERE` antes que el `SELECT`. Al usar `PARTITION BY`, obtenemos un ranking independiente por colonia, y el `ROW_NUMBER` asegura un conteo secuencial único (1, 2, 3) sin importar empates en los valores.

```sql
SELECT*
FROM (
SELECT
    colonia,
    tipo,
    valor,
ROW_NUMBER()OVER (
PARTITION BY colonia
ORDER BY valor DESC
    )AS ranking
FROM app.datos
) t
WHERE ranking<=3;
```

Data Output | Messages | Notifications

Show

	colonia character varying	tipo character varying	valor numeric (12,2)	ranking bigint
1	Ajusco	Galpón/Depósito/Almacén	42000.00	1
2	Ajusco	Departamento	28000.00	2
3	Ajusco	Casa de Condominio	26250.00	3
4	Álvaro Obregón	Casa	5250.00	1
5	Álvaro Obregón	Departamento	4550.00	2
6	Álvaro Obregón	Departamento	4550.00	3
7	Arboledas	Tienda/Salón	42000.00	1
8	Arboledas	Galpón/Depósito/Almacén	30800.00	2

🔷 BLOQUE 4 – Optimización y buenas prácticas

Ejercicio:

Crear índice

```
CREATE INDEX idx_datos_colonia ON app.datos(colonia);

--CREATE INDEX → le dice a PostgreSQL que cree una estructura auxiliar
--idx_datos_colonia → nombre del índice
--ON app.datos(colonia) → el índice se crea sobre la columna colonia
```

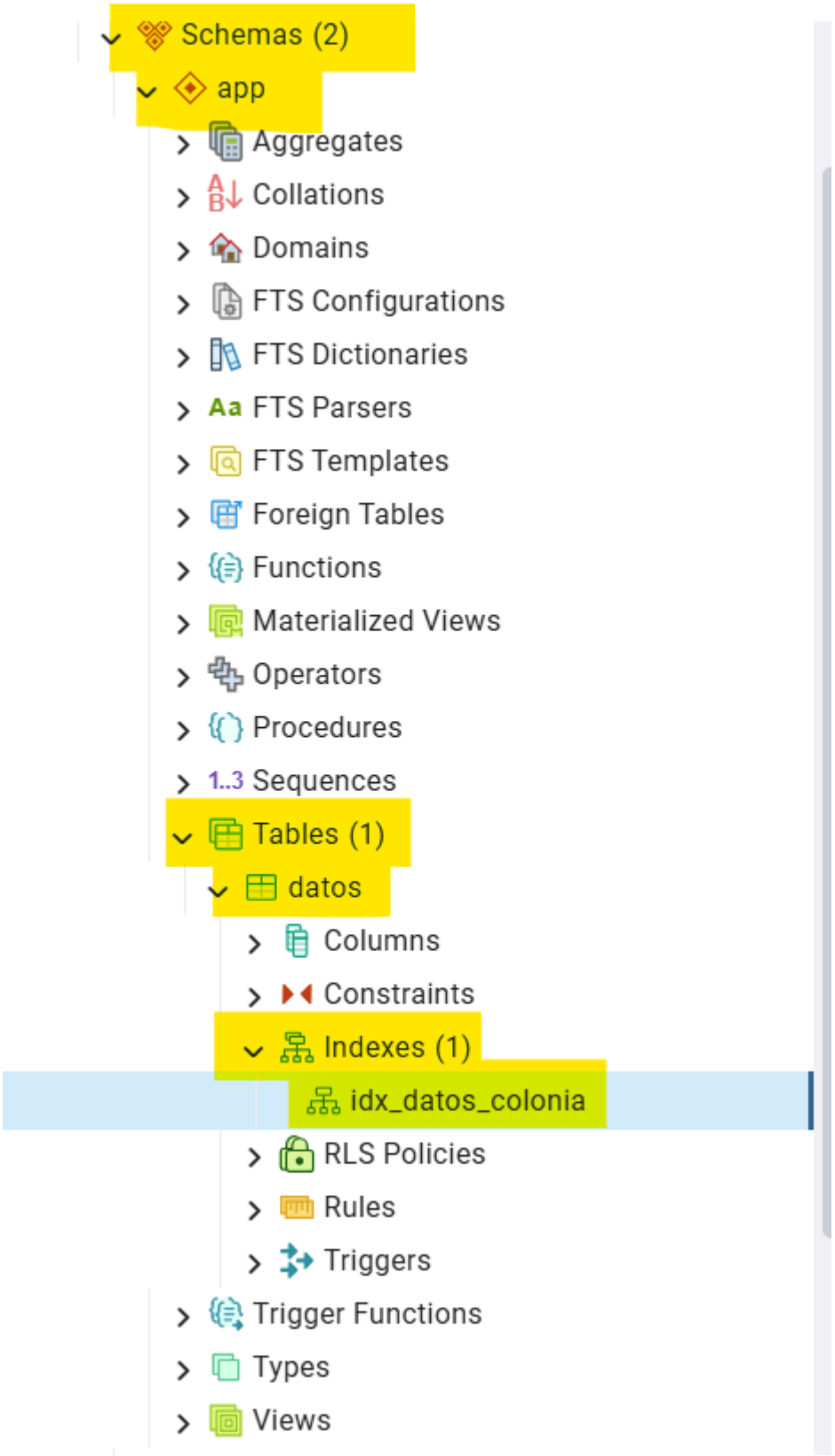
Un **índice** en PostgreSQL sirve para **acelerar las consultas** que buscan, filtran u ordenan datos por una columna específica.

Antes del índice:

- PostgreSQL hace **Sequential Scan** (recorre toda la tabla)

Después del índice:

- PostgreSQL puede hacer **Index Scan**
- Mucho más rápido, especialmente con tablas grandes


Schemas (2)
app
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (1)
datos
Columns
Constraints
Indexes (1)
idx_datos_colonia
RLS Policies
Rules
Triggers
Trigger Functions
Types
Views

```
EXPLAIN ANALYZE
SELECT *
FROM app.datos
WHERE colonia = 'Narvarte';
```

`EXPLAIN ANALYZE` ejecuta la consulta y muestra:

- Cómo PostgreSQL decidió resolver la consulta
- Cuánto tiempo real tomó para PosgreSQL hace la consulta

	QUERY PLAN text
1	Bitmap Heap Scan on datos (cost=5.26..221.69 rows=126 width=98) (actual time=0.421..0.777 rows=1380 loops=1)
2	Recheck Cond: ((colonia)::text = 'Narvarte'::text)
3	Heap Blocks: exact=231
4	-> Bitmap Index Scan on idx_datos_colonia (cost=0.00..5.23 rows=126 width=0) (actual time=0.394..0.394 rows=1380 loo…
5	Index Cond: ((colonia)::text = 'Narvarte'::text)
6	Planning Time: 1.775 ms
7	Execution Time: 1.437 ms

- [x] ~~Estudio del tema~~
- [x] ~~Ejecución de scripts~~
- [x] ~~Taller práctico~~
- [x] ~~Evidencia guardada~~
- [x] ~~Notas técnicas escritas~~