Contenido

[¿Qué es SQL y para qué se utiliza en ciencia de datos?✔️ 3](#_Toc207350009)

[¿Cuál es la diferencia entre WHERE y HAVING? ✔️ 3](#_Toc207350010)

[Ejercicios con where 6](#_Toc207350011)

[Ejercicios con having 7](#_Toc207350012)

[¿Qué diferencia hay entre INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL OUTER JOIN? ✔️ 9](#_Toc207350013)

[¿Cuál es la diferencia entre COUNT(\*) y COUNT(columna)? ✔️ 9](#_Toc207350014)

[¿Qué significa NULL en SQL y cómo se maneja? ✔️ 11](#_Toc207350015)

[¿Qué es una clave primaria y una clave foránea? ✔️ 11](#_Toc207350016)

[¿Qué hace la cláusula GROUP BY? ✔️ 12](#_Toc207350017)

[¿Cuál es la diferencia entre DISTINCT y GROUP BY? ✔️ 13](#_Toc207350018)

[¿Qué es una subconsulta y cuándo se usaría?\* 14](#_Toc207350019)

[¿Qué diferencia hay entre UNION y UNION ALL?✔️ 17](#_Toc207350020)

[**🟢 Básico (1-10)** 19](#_Toc207350021)

[¿Qué tipos de JOIN existen?✔️ 19](#_Toc207350022)

[¿Qué hace el comando GROUP BY?✔️ 20](#_Toc207350023)

[¿Cuál es la diferencia entre NULL, 0 y una cadena vacía ('')?✔️ 20](#_Toc207350024)

[¿Cómo eliminarías registros duplicados de una tabla?✔️ 21](#_Toc207350025)

[¿Qué función cumple DISTINCT?✔️ 21](#_Toc207350026)

[¿Qué es una subconsulta (subquery) y cómo se diferencia de una JOIN?\* 22](#_Toc207350027)

[**🟡 Intermedio (11–20)** 22](#_Toc207350028)

[¿Cuál es la diferencia entre INNER JOIN, RIGHT JOIN y LEFT JOIN en la práctica?✔️ 22](#_Toc207350029)

[¿Qué son las funciones de agregación? Menciona al menos cinco ejemplos.✔️ 24](#_Toc207350030)

[¿Cuál es la diferencia entre UNION y UNION ALL?✔️ 26](#_Toc207350031)

[Diferencia entre unique y distinc ✔️ 26](#_Toc207350032)

[¿Cómo actualizarías múltiples filas en una tabla con una sola sentencia UPDATE?❕ 27](#_Toc207350033)

[¿Qué es una tabla derivada y cómo se usa?✔️ 27](#_Toc207350034)

[¿Cómo contarías el número de empleados por departamento, pero solo de los que ganan más de $1000?✔️ 29](#_Toc207350035)

[¿Qué significa normalización de datos y por qué es importante?✔️ 29](#_Toc207350036)

[¿Qué es un índice y cómo afecta el rendimiento de las consultas?✔️ 30](#_Toc207350037)

[¿Cómo puede la optimización de los índices de una base de datos mejorar el rendimiento?✔️ 30](#_Toc207350038)

[¿Cuál es el orden de ejecución de una consulta SQL?✔️ 34](#_Toc207350039)

[Cómo pensar una consulta SQL 36](#_Toc207350040)

[**🔵 Avanzado (21–30)❕** 37](#_Toc207350041)

[¿Qué es una CTE (Common Table Expression) y cuándo usarla? 37](#_Toc207350042)

[¿Cómo calcularías un promedio móvil con SQL? 37](#_Toc207350043)

[¿Qué diferencias hay entre EXISTS, IN y NOT IN? 37](#_Toc207350044)

[¿Cómo manejarías valores NULL en funciones agregadas? 37](#_Toc207350045)

[¿Qué son las funciones analíticas y cómo se diferencian de las agregadas? 37](#_Toc207350046)

[¿Cómo detectarías outliers usando SQL? 37](#_Toc207350047)

[¿Qué es una transacción y cuáles son sus propiedades (ACID)? 37](#_Toc207350048)

[¿Qué es una vista (view) y cuáles son sus ventajas y desventajas? 37](#_Toc207350049)

[¿Cómo optimizarías una consulta lenta? 38](#_Toc207350050)

[¿Cómo harías una consulta para encontrar la segunda fila más alta en una tabla de ventas? 38](#_Toc207350051)

[Enfoque en ciencia de datos 38](#_Toc207350052)

[¿Cómo identificarías a los clientes más valiosos para un negocio usando SQL o Python? ✔️ 38](#_Toc207350053)

[Si las ventas de un producto disminuyen, ¿cómo investigarías las causas usando datos? ✔️ 39](#_Toc207350054)

[¿Cómo construirías un análisis de cohortes en SQL? ¿Qué información te aporta?❕ 39](#_Toc207350055)

[Qué son las KPIs ✔️ 39](#_Toc207350056)

[¿Cómo calcularías la tasa de retención de usuarios mes a mes con SQL?✔️ 40](#_Toc207350057)

[¿Cómo agruparías datos por semana, mes o trimestre en SQL? ¿Y en Pandas? 40](#_Toc207350058)

[¿Cómo calcularías el crecimiento porcentual mes a mes de una métrica? 40](#_Toc207350059)

[¿Qué es un “lag” en series de tiempo y cómo lo aplicarías en una base de datos? 40](#_Toc207350060)

[¿Cómo identificarías una tendencia estacional en una serie temporal? 40](#_Toc207350061)

[¿Qué harías si te encuentras con timestamps en diferentes zonas horarias? 40](#_Toc207350062)

[¿Cómo detectarías outliers en una variable numérica usando SQL o Python? 41](#_Toc207350063)

[¿Qué pasos seguirías si una tabla tiene muchas columnas con más del 50% de valores nulos? 41](#_Toc207350064)

[¿Cómo tratarías datos categóricos con errores tipográficos o formatos inconsistentes (como “Male”, “M”, “male”)? 41](#_Toc207350065)

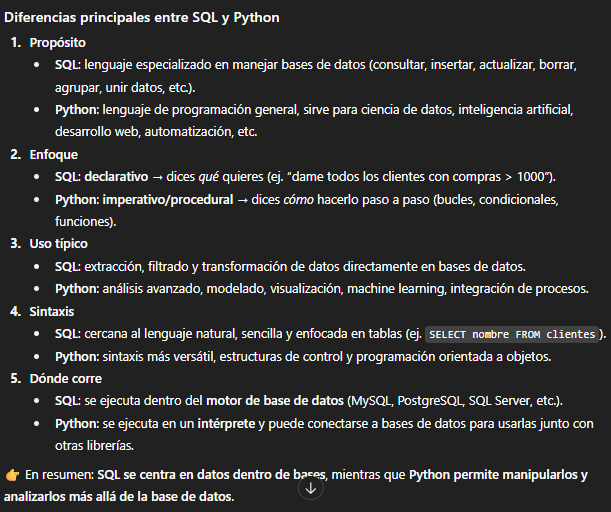
[Te entregan un dataset de usuarios y eventos. ¿Cómo calcularías la conversión desde el registro hasta una compra? 41](#_Toc207350066)

[Tienes un dataset con 100 columnas. ¿Cómo identificarías las variables más relevantes para predecir una salida (target)? 41](#_Toc207350067)

Promt: Genera una tabla con 10 observaciones y dame un ejercicio nivel Y, K, L para practicar Z en SQL. Dame también la salida esperada

# ¿Qué es SQL y para qué se utiliza en ciencia de datos?✔️

SQL es un lenguaje de consulta estructurado, **no un lenguaje de programación general**, especializado en el almacenamiento y consulta de bases de datos relacionales. Existen diferentes motores como MySQL, PostgreSQL, SQL server, etc. En la ciencia de datos lo utilizamos para almacenar datos, estructurarlos, limpiarlos y poder acceder a ellos de forma eficiente, escalable y segura



sql es un lenguaje declarativo (le dices qué hacer) y python es imperativo (le dices cómo hacerlo)

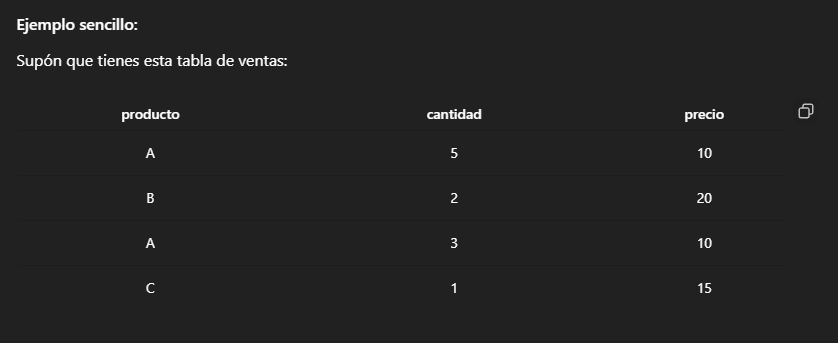
# ¿Cuál es la diferencia entre WHERE y HAVING? ✔️

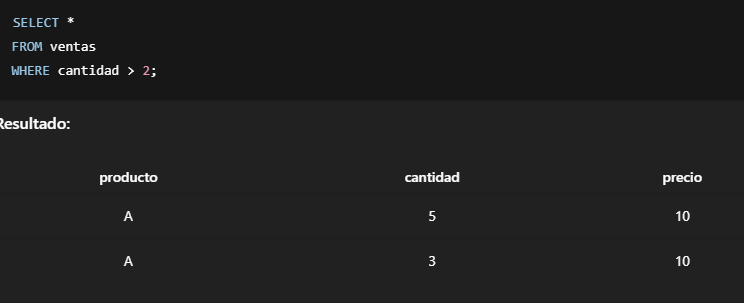
 **WHERE** se usa para **filtrar filas** *antes* de que se agrupen los datos. Es decir, filtra directamente sobre las filas de la tabla.

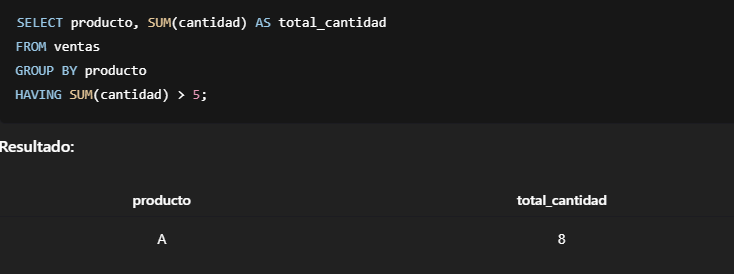
 **HAVING** se usa para **filtrar grupos** *después* de hacer una agregación con GROUP BY. Es decir, filtra sobre el resultado de las agrupaciones.

Resumen: where se aplica sobre la tabla bruta having se aplica tras una agrupación

En el where No se pueden escribir funciones de agregación pero en el having si

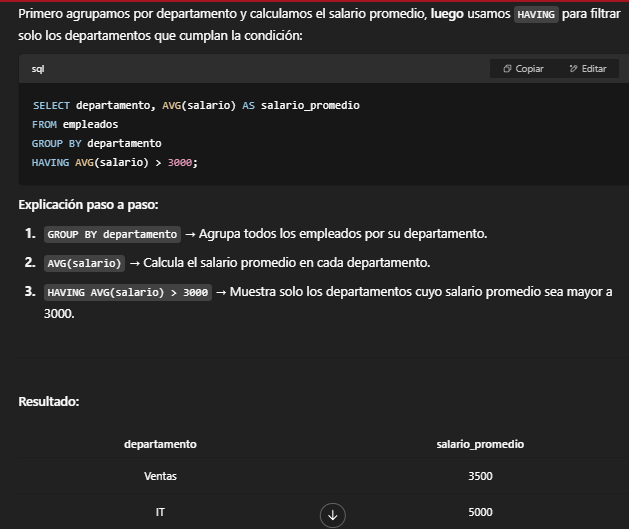






Veamos otro ejemplo





En conclusión podemos decir que el having Es una filtración de la filtración agrupada previamente

* En el where no podemos poner funciones de agregación
* En el having si podemos poner funciones de agregacion

## Ejercicios con where



🟢 Ejercicio Fácil

**Pregunta:**  
Selecciona todos los clientes que vivan en Guadalajara.

Select c.nombre, c.ciudad

From clientes c

Where c.ciudad = “Guadalajara”;

🟡 Ejercicio Intermedio

**Pregunta:**  
Selecciona los clientes que tengan un saldo mayor a 2000 **y** sean menores de 40 años.

Select c.nombre

From clients c

Where c.saldo > 2000 and c.edad < 40;

🔴 Ejercicio Difícil

**Pregunta:**  
Selecciona los clientes registrados después del 1 de enero de 2022 que **no** sean de Ciudad de México **y** cuyo saldo sea inferior a 1000.

SELECT \* FROM clientes

WHERE fecha\_registro > '2022-01-01'

AND ciudad <> 'Ciudad de México'

AND saldo < 1000;

## Ejercicios con having

🟢 Ejercicio Fácil

**Pregunta:**  
Agrupa por ciudad y muestra solo aquellas donde haya **más de 1 cliente**.

Select c.ciudad, count(\*) as total\_clientes

From clientes c

Group by ciudad

having count(\*) > 1;



“Selecciona la columna ciudad de la tabla clientes y cuenta cuántas veces aparece cada ciudad, es decir, cuántos clientes hay por ciudad. Esta cuenta se almacena en una columna temporal llamada total\_clientes. Luego, se agrupan las filas según los valores únicos de ciudad usando GROUP BY. Finalmente, el filtro HAVING COUNT(\*) > 1 asegura que solo se incluyan en el resultado aquellas ciudades que tienen **más de un cliente** asociado.”

🟡 Ejercicio Intermedio

**Pregunta:**  
Agrupa por ciudad y muestra aquellas donde el **saldo promedio de los clientes supere los 2000**.

Select c.ciudad, avg(saldo) as salario\_promedio

From clientes c

Group by ciudad

Having avg(salario) > 2000;

🔴 Ejercicio Difícil❕

**Pregunta:**  
Agrupa por ciudad y muestra aquellas donde el **número de clientes menores de 30 años sea al menos 2**.

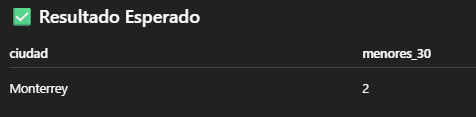
SELECT ciudad,

COUNT(CASE WHEN edad < 30 THEN 1 END) AS menores\_30

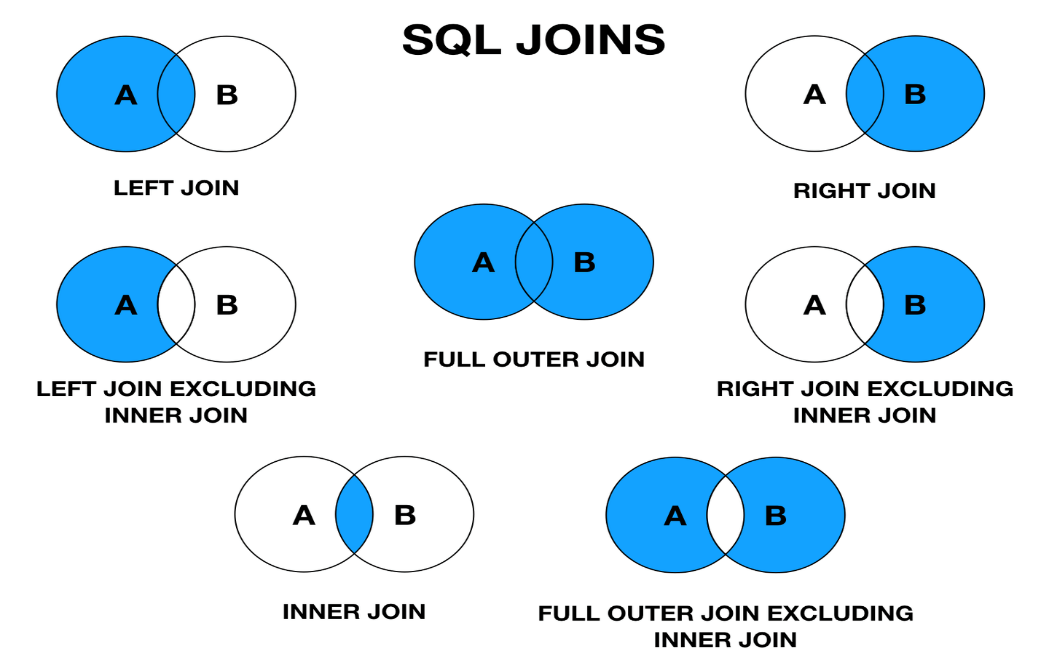
FROM clientes

GROUP BY ciudad

HAVING COUNT(CASE WHEN edad < 30 THEN 1 END) >= 2;



# ¿Qué diferencia hay entre INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL OUTER JOIN? ✔️



Este tipo de referencias solo existen cuando tenemos tablas relacionadas (Puede ser dos o más tablas)

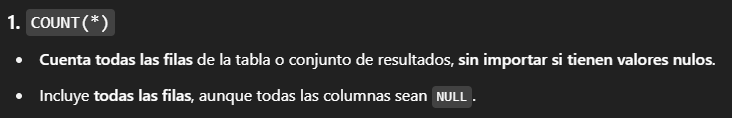
Inner join: Sólo se trae la información que comparte la tabla A y la tabla B. Es decir, trae la intersección de la tabla A con la tabla B

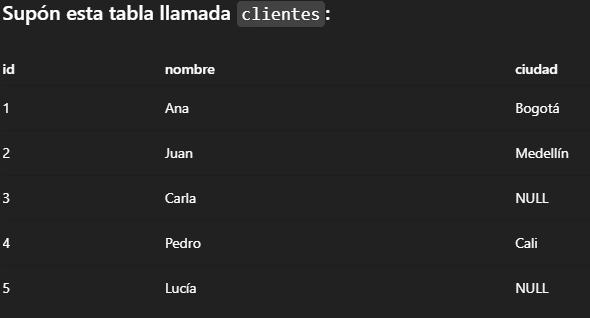
Left join: Trae a todas las observaciones de la tabla A y además la intersección con la tabla B

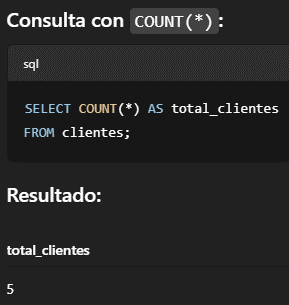
Right join: Trae todas las observaciones de la tabla B además de la intersección con la tabla A

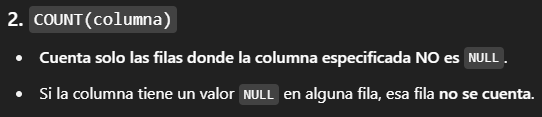
Full outer join: Trae todos los registros de la tabla A y la tabla B

# ¿Cuál es la diferencia entre COUNT(\*) y COUNT(columna)? ✔️

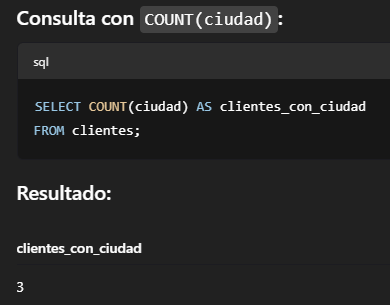








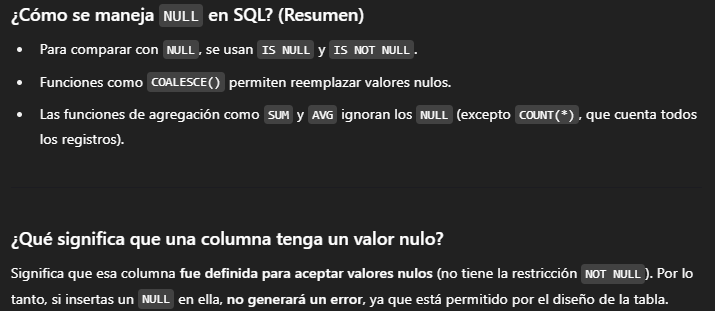
Consideremos la misma tabla que anteriormente



Es decir el count(columna) Ignora los valores nulos no los contabiliza y sin embargo el count (\*) si los contabiliza

# ¿Qué significa NULL en SQL y cómo se maneja? ✔️

Null significa que la tabla en cuastion si acepta valores nulos y por ende, la observación de esa columna no contiene valor alguno. Es decir, representa un **valor desconocido, ausente o no aplicable**. No significa "cero" ni una cadena vacía — simplemente indica que **no hay ningún valor asignado** en esa celda.



# ¿Qué es una clave primaria y una clave foránea? ✔️

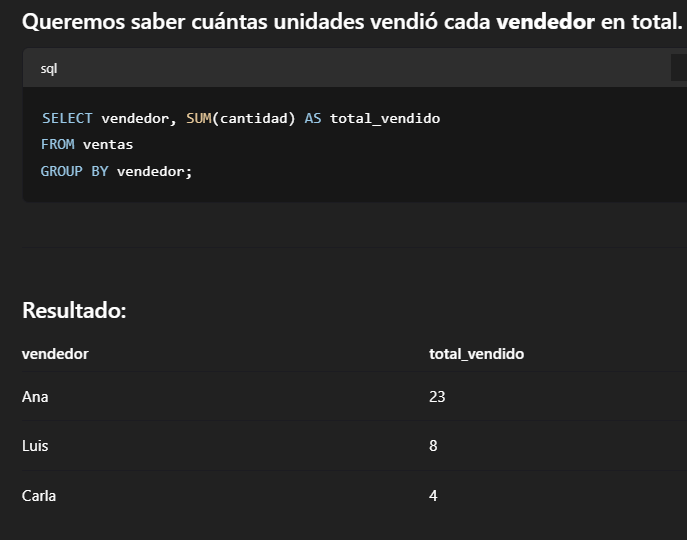
La clave primaria es un campo (o conjunto de campos) en una tabla que identifica de forma única a cada fila o registro. No puede contener valores nulos ni duplicados. Cada tabla debe tener una única clave primaria.

La clave foránea es un campo (o conjunto de campos) en una tabla que hace referencia a la clave primaria de otra tabla. Sirve para establecer relaciones entre tablas y mantener la integridad referencial de los datos.

# ¿Qué hace la cláusula GROUP BY? ✔️

La cláusula GROUP BY en SQL **agrupa las filas que tienen los mismos valores en una o más columnas** en conjuntos, para que se pueda aplicar funciones de agregación (como SUM, COUNT, AVG, etc.) a cada grupo.



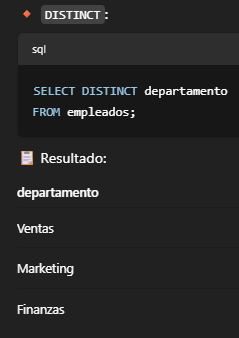


# ¿Cuál es la diferencia entre DISTINCT y GROUP BY? ✔️

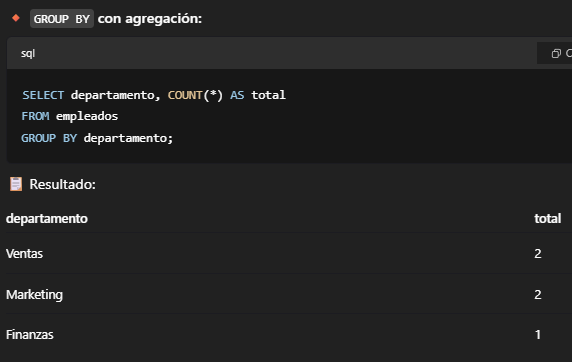
* DISTINCT Se usa para **eliminar filas duplicadas** en los resultados y mostrar solo combinaciones únicas de columnas.
* Group by Se usa para **agrupar filas** que tienen los mismos valores en columnas específicas, **para aplicar funciones de agregación** como SUM(), COUNT(), AVG(), etc.

Para comprenderlos veamos un ejemplo práctico





es decir, el DISTINCT Sirve para saber las categorías de una columna (No ignoran los valores nulos los cuenta como una categoría más) y además elimina los duplicados en la consulta



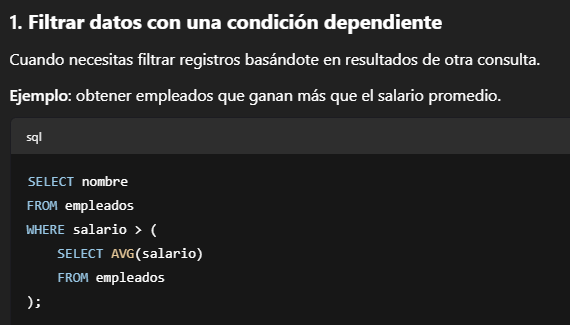
# ¿Qué es una subconsulta y cuándo se usaría?\*

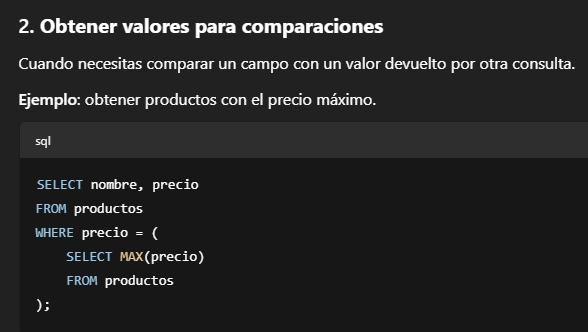
En SQL, un subquery (subconsulta) es una consulta anidada dentro de otra consulta. La subconsulta se ejecuta primero y su resultado se utiliza por la consulta externa. Los subqueries son útiles para realizar consultas más complejas y obtener datos específicos que dependen de resultados intermedios.

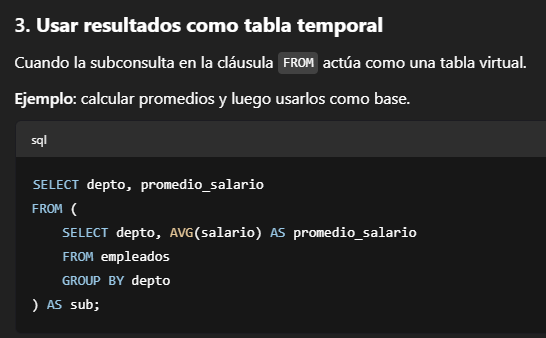
Propósito y Ventajas de los Subqueries

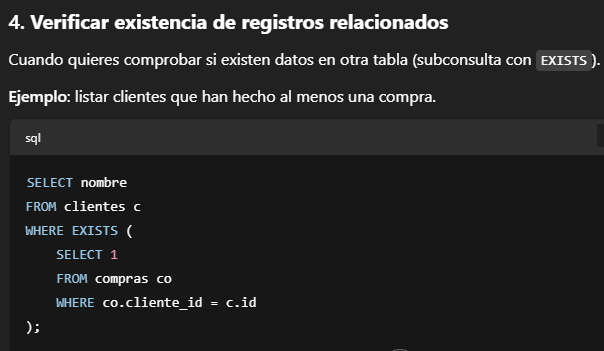
1. **Modularidad**: Facilitan la descomposición de una consulta compleja en partes más manejables.
2. **Reutilización**: Permiten el uso de resultados intermedios en varias partes de la consulta principal.
3. **Flexibilidad**: Ayudan a realizar consultas que serían difíciles o menos eficientes de realizar de otra manera.
4. **Claridad**: Pueden mejorar la legibilidad de una consulta al aislar partes específicas de la lógica

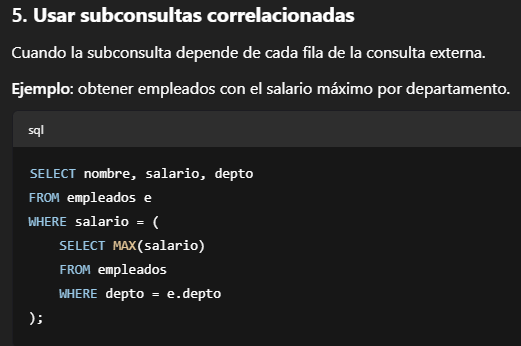
Las subconsultas se usan en SQL cuando necesitas resolver una parte de la consulta dependiendo de otro conjunto de datos





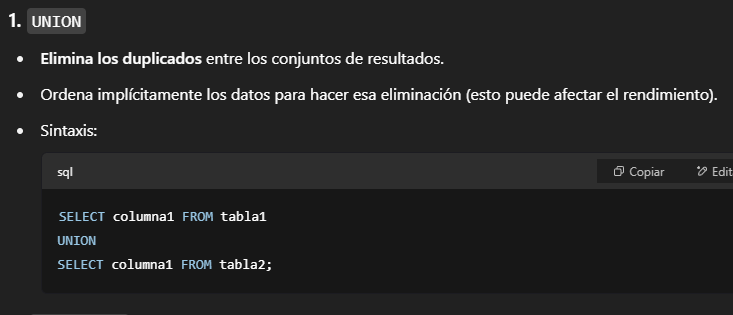


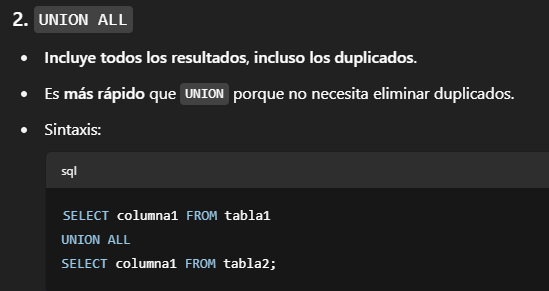




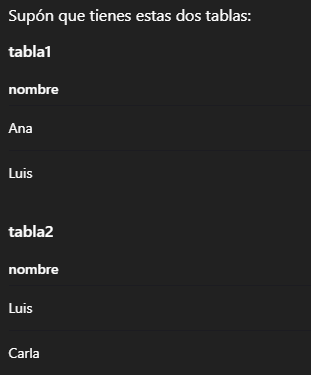
# ¿Qué diferencia hay entre UNION y UNION ALL?✔️

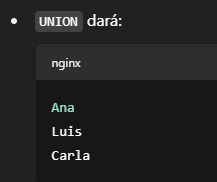
En SQL, tanto UNION como UNION ALL se utilizan para **combinar los resultados de dos o más consultas SELECT**, pero hay una diferencia clave entre ellos:

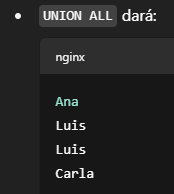




Ahora veamos un ejemplo práctico





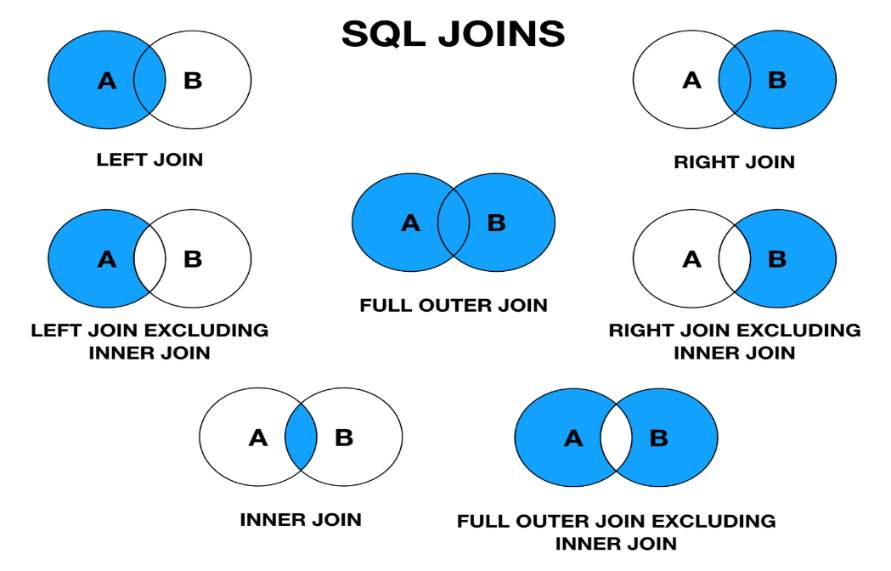


la condición para que union o union all funcione es que deben ser los mismos campos o columnas en número y forma (tipo de dato, orden lógico de las columnas)

**🟢 Básico (1-10)**

# ¿Qué tipos de JOIN existen?✔️

Existen multiples tipos de join, pero los mas usuales son:



Supongamos que tenemos una tabla A y una tabla B, donde la tabla a es aquella que se pone antes del join y la tabla B es la que se pone junto al Join (en el codigo)

* Inner join: Trae la intersección de la tabla A con la tabla B
* Ful outer join excluiding inner join: Trae la table A y la table B excluyendo la intersección de éstas
* Left join: Trae la table A y la intersección con la table B
* Left join excluiding inner join: Trae la table A y excluye la intersección con la table B
* Right join: Devuelve **todas las filas de la tabla B** (derecha) y las filas coincidentes de la tabla A. Si no hay coincidencia, A tendrá valores NULL.
* Right joing excluiding inner join: Devuelve las filas de la tabla B **que no tienen coincidencia** en la tabla A.
* Full outer join: Trae la table A y la table B así como sus intersecciones (las intersecciones se unifican, no se suman)

# ¿Qué hace el comando GROUP BY?✔️

El comando group by agrupa categorías de una columna según una especificación y le aplica funciones de agregacion

# ¿Cuál es la diferencia entre NULL, 0 y una cadena vacía ('')?✔️

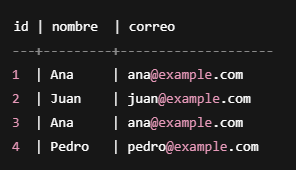
Null significa que no hay valor asignado en la celda

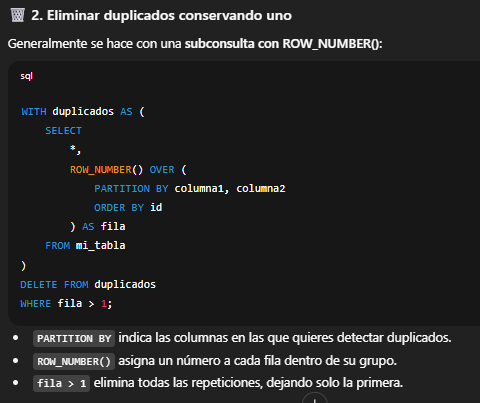
Cero significa que existe un valor en la celda y es el cero (ya sea como int o varchar)

Una cadena vacia significa que hay un string existente en la columna (Es decir el tipo de dato es de texto ya sea varchar o char o cualquier otro compatible)

# ¿Cómo eliminarías registros duplicados de una tabla?✔️

Haría una consulta con distinct con distinct (no elimina nada pero se puede descargar la tabla y subirla a la base de datos)





# ¿Qué función cumple DISTINCT?✔️

**DISTINCT** es una cláusula en SQL que **elimina filas duplicadas** del resultado de una consulta, mostrando **solo valores únicos** de una o más columnas. Viéndolo de una Forma distinta también sirve para saber las categorías de una columna incluyendo los valores nulos es decir los valores nulos los toma como una categoría

# ¿Qué es una subconsulta (subquery) y cómo se diferencia de una JOIN?\*

Una subconsulta es una consulta anidada y un join es una consulta combinada donde se involucra dos o más tablas relacionadas

La subconsulta es una consulta dentro de la consulta principal

**🟡 Intermedio (11–20)**

# ¿Cuál es la diferencia entre INNER JOIN, RIGHT JOIN y LEFT JOIN en la práctica?✔️

* El inner join tiene la estructura siguiente:

select columnas

from tabla A

inner join table B

on id = id;

En particular aquí da igual a disposición de las tablas ya que la naturaleza del inner join es la intersección

* El left join Tienen la estructura siguiente:

select columnas

from tabla A (tabla izquierda)

left join tabla B (tabla derecha)

on id = id;

En esta parte es obligatorio colocar la tabla A Junto al from y la tabla b junto al Join.

* En el right join que la tabla A esta junto al join y la B junto al from

select columnas

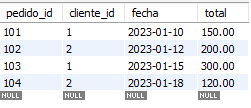
from tabla b (tabla izquierda)

right join tabla a (tabla derecha)

on id = id;

considere las siguientes dos tablas:





* El left join:

Select c.cliente\_id, c.nombre, c.ciudad, p.fecha, p.total

From clientes c

Left join pedidos p

On c.cliente\_id = p.cliente\_id;



* Misma consulta con right join

Select c.cliente\_id, c.nombre, c.ciudad, p.fecha, p.total

From pedidos p

right join clientes c

On c.cliente\_id = p.cliente\_id;



* Otra consulta con right join

Select c.cliente\_id, c.nombre, c.ciudad, p.fecha, p.total

From clientes c

right join pedidos p

On c.cliente\_id = p.cliente\_id;



* Misma consulta, pero con left join

Select c.cliente\_id, c.nombre, c.ciudad, p.fecha, p.total

From pedidos p

left join clientes c

On c.cliente\_id = p.cliente\_id;



En SQL, **LEFT JOIN y RIGHT JOIN son simétricos**:

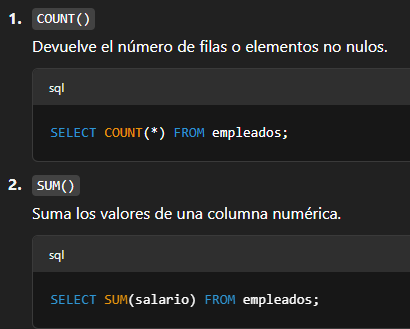
* A LEFT JOIN B ON ... ≡ B RIGHT JOIN A ON ...

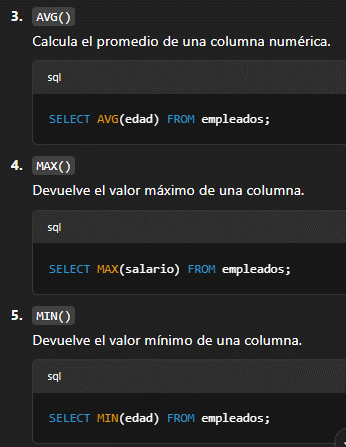
Siempre que pongas la misma condición y cambies correctamente el orden de las tablas, el resultado es el mismo.

En la práctica es más utilizado left join

# ¿Qué son las funciones de agregación? Menciona al menos cinco ejemplos.✔️

Las **funciones de agregación en SQL** son funciones que **operan sobre un conjunto de valores** y **devuelven un único valor de resumen**. Se utilizan comúnmente en combinación con GROUP BY para agrupar datos y obtener estadísticas o resúmenes.





Algunas otras comunes:

 **VARIANCE()** o **VAR\_SAMP()** → Calcula la varianza muestral.

 **VAR\_POP()** → Varianza poblacional.

 **STDDEV()** o **STDDEV\_SAMP()** → Desviación estándar muestral.

 **STDDEV\_POP()** → Desviación estándar poblacional.

 **MEDIAN()** (disponible en algunos motores como Oracle y PostgreSQL con extensiones) → Devuelve la mediana.

 **MODE()** (en Oracle, PostgreSQL con extensiones) → Valor más frecuente.

 **GROUP\_CONCAT()** (MySQL) o **STRING\_AGG()** (PostgreSQL/SQL Server) → Concatenar valores en un solo string.

# ¿Cuál es la diferencia entre UNION y UNION ALL?✔️

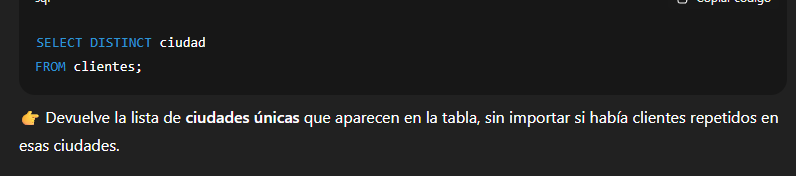
Son cláusulas que permiten unir dos o más consultas. Union permite unir dos consultas ignorando valores duplicados (los elimina) y unión all no ignora duplicados (trae todo)

# Diferencia entre unique y distinc ✔️

Te explico la diferencia entre **UNIQUE** y **DISTINCT**, porque aunque parecen lo mismo, en SQL (y en general en bases de datos) no lo son

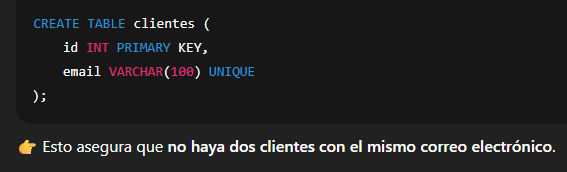
🔹 DISTINCT

* **Es una cláusula** que se usa en consultas SELECT.
* **Sirve para eliminar duplicados** en los resultados de una consulta.
* Se aplica solo al **resultado de la consulta**, no cambia la estructura de la tabla ni sus datos almacenados.



UNIQUE

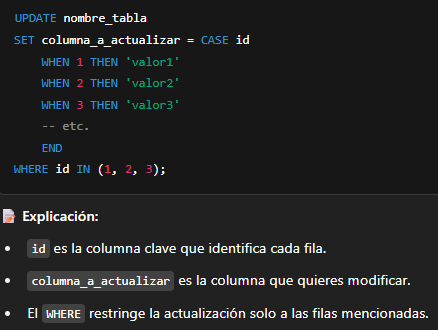
* **Es una restricción (constraint)** que se aplica cuando creas o alteras una tabla.
* **Sirve para asegurar que los valores de una columna o combinación de columnas sean únicos dentro de la tabla** (no permite duplicados en la base de datos).
* Afecta la **estructura de la tabla** y garantiza integridad de datos.



**Resumiendo**:

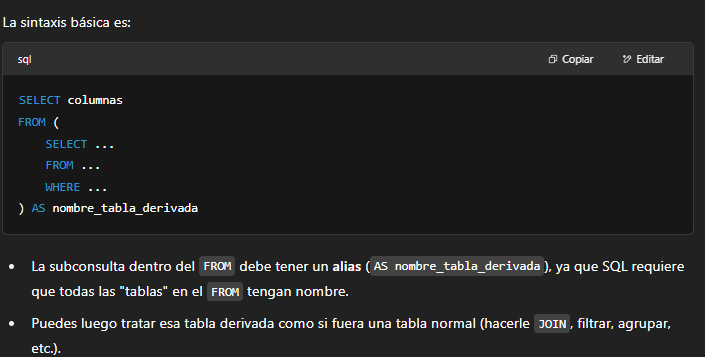
* DISTINCT → elimina duplicados en el **resultado de una consulta**.
* UNIQUE → evita duplicados en la **estructura/almacenamiento de la tabla**.

# ¿Cómo actualizarías múltiples filas en una tabla con una sola sentencia UPDATE?❕

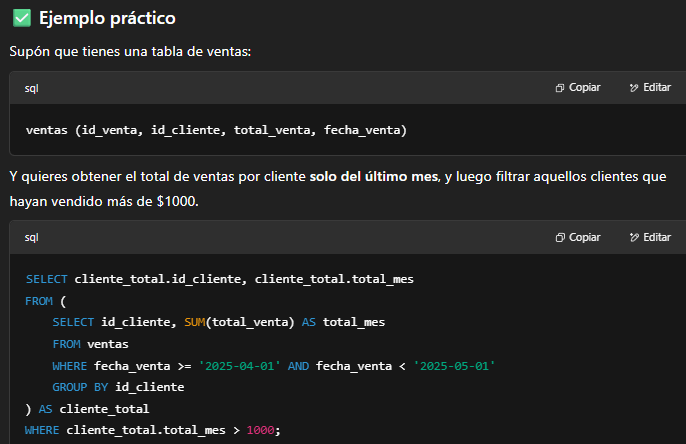


# ¿Qué es una tabla derivada y cómo se usa?✔️

Una **tabla derivada** en SQL es una **subconsulta que se coloca en la cláusula FROM** de una instrucción SELECT. Actúa como una tabla temporal que solo existe durante la ejecución de esa consulta.



Veamos un ejemplo para saber cómo funciona las tablas derivadas en una subconsulta:



# ¿Cómo contarías el número de empleados por departamento, pero solo de los que ganan más de $1000?✔️

SELECT departamento, COUNT(\*) AS num\_empleados

FROM TABLA

WHERE salario > 1000

GROUP BY departamento;

# ¿Qué significa normalización de datos y por qué es importante?✔️

Normalizar una base de datos significa estructurar de forma lógica y correcta todas las tablas para que no haya errores lógicos al momento de hacer consultas o cualquier otro fin. Es decir, significa ordenar las tablas a nivel lógico o relacional.

Las ventajas que tiene la normalización:

 Eliminar redundancias (datos repetidos).

 Evitar inconsistencias.

 Mejorar la integridad de los datos.

 Facilitar el mantenimiento a largo plazo.

Esto se hace mediante las 3 formas normales (o a veces 5 formas normales). Se debe seguir un proceso metodológico para poder normalizar correctamente una base de datos

Cabe señalar que en ciencia de datos cuando usamos el término normalización nos referimos a un concepto no tan técnico. Nos referimos al hecho de escalar o dejar la base de datos limpia para poder hacer manipulaciones estadísticas correctamente

# ¿Qué es un índice y cómo afecta el rendimiento de las consultas?✔️

# ¿Cómo puede la optimización de los índices de una base de datos mejorar el rendimiento?✔️

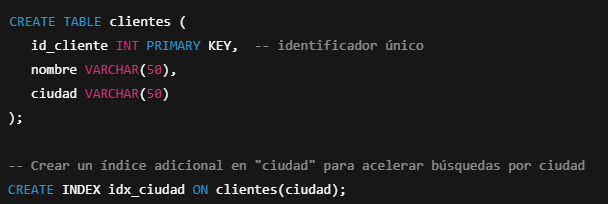
¿Qué es un índice?

¡¡¡¡Un índice no es lo mismo que un identificador único (clave primaria)!!!!

* La **clave primaria en una tabla sí es un identificador único**, pero **no todos los índices son identificadores únicos**.
* Un índice es una estructura de datos auxiliar que permite acceder a los registros de una tabla más rápido sin necesidad de recorrer fila por fila. No necesariamente es único (aunque puede serlo). El **índice en una base de datos es invisible para ti como usuario**.
  + No aparece como una columna en la tabla.
  + No lo consultas directamente (a menos que uses comandos especiales para inspeccionarlo).
  + Su única función es ser una **estructura auxiliar interna** que acelera las búsquedas, filtros, uniones (JOIN), y ordenamientos (ORDER BY).

Entonces:

* **Clave primaria** = valor único para identificar filas.
* **Índice** = estructura que acelera las consultas (puede estar en columnas únicas o no).
* Por convención, **las claves primarias siempre tienen un índice asociado automáticamente** (porque es muy común buscarlas).



**Ejemplo**

1. Creamos una tabla e insertamos datos

CREATE TABLE clientes (

id\_cliente INT PRIMARY KEY, -- clave primaria (única)

nombre VARCHAR(50),

ciudad VARCHAR(50)

);

INSERT INTO clientes (id\_cliente, nombre, ciudad) VALUES

(1, 'Juan Pérez', 'CDMX'),

(2, 'Ana López', 'Monterrey'),

(3, 'Carlos Díaz', 'CDMX'),

(4, 'Ana López', 'Guadalajara'),

(5, 'Lucía Torres', 'CDMX');

Observa que:

* El id\_cliente es único (1, 2, 3, 4, 5).
* El nombre **no es único** (hay dos "Ana López").

La tabla completa es:



1. Buscamos por clave primaria

SELECT \* FROM clientes WHERE id\_cliente = 2;



Como Podemos observer nos arrojó solo el registro que lleva por clave primaria el número dos

1. Buscamos por columna sin índice

SELECT \* FROM clientes WHERE nombre = 'Ana López';



El motor de la base debe revisar fila por fila (si no hay índice en nombre).

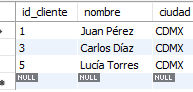
1. Crear un índice en la columna ciudad

CREATE INDEX idx\_ciudad ON clientes(ciudad);

Este índice **no garantiza unicidad**, solo acelera búsquedas por ciudad.

1. Buscamos usando el índice

SELECT \* FROM clientes WHERE ciudad = 'CDMX';



Gracias al índice, la búsqueda es rápida (aunque los valores no son únicos).

1. Intentar romper la clave primaria

INSERT INTO clientes (id\_cliente, nombre, ciudad)

VALUES (2, 'Pedro Morales', 'Tijuana');

 ❌ Error: id\_cliente ya existe.

 Porque la **clave primaria no permite duplicados**.

**Conclusión práctica**

* **Clave primaria (id\_cliente)** → identificador único, automáticamente indexado.
* **Índice (ciudad)** → atajo de búsqueda, pero puede tener valores repetidos.

**Conclusion**

El índice no se ve como datos de la tabla, es una estructura oculta y auxiliar que solo el motor usa para acelerar operaciones.

¿Cómo mejoran los índices el rendimiento?

La clave está en entender que **los índices cambian la forma en que la base de datos busca los datos**.

* Sin índices

Imagina una tabla con **10 millones de registros**.

Si hacemos:

SELECT \* FROM clientes WHERE nombre = 'Ana López';

El motor debe revisar **fila por fila** (búsqueda secuencial, full table scan).  
En el peor caso, revisa los 10 millones de registros → **muy lento**.

* Con índices

Si existe un índice en la columna nombre, la base de datos hace algo como:

* Va al índice, que está ordenado (como un diccionario o un árbol).
* Encuentra rápidamente “Ana López”.
* Desde ahí “salta” directamente a la fila o filas que coinciden.

➡️ Ya no revisa todo, solo un subconjunto muy pequeño.

 **Sin índice**: buscar un registro entre 10,000,000 → tiempo proporcional a 10,000,000 operaciones.

 **Con índice tipo B-Tree**: se reduce a **log₂(10,000,000) ≈ 23 pasos** para encontrarlo.

Casos donde mejoran el rendimiento

* **Filtros en consultas** (WHERE ciudad = 'CDMX').
* **Búsquedas por rangos** (BETWEEN, >, <).
* **Uniones** (JOIN) cuando las columnas involucradas están indexadas.
* **Ordenamientos y agrupaciones** (ORDER BY, GROUP BY) porque el índice ya mantiene un orden.

Pero ojo:

* Los índices **no son gratis**:
  + Aceleran las **lecturas** (SELECT).
  + Pero hacen más costosas las **escrituras** (INSERT, UPDATE, DELETE), porque también hay que actualizar el índice.
* Por eso, no conviene indexar todas las columnas, solo las que realmente se usan mucho en consultas.

En resumen:  
**Los índices mejoran el rendimiento porque evitan que la base de datos recorra toda la tabla registro por registro. En vez de eso, usan estructuras ordenadas (como árboles o tablas hash) que permiten saltar directamente a los datos buscados.**

Idea intuitiva a recordar:

- ayudan a mejorar el rendimiento de una consulta porque disminuye el numero de busquedas segun la naturaleza de la consulta. Esto lo hacen gracias a una "caja negra" (que no entiendo, pero lo hacen)

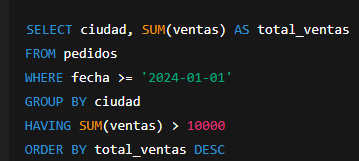
- hay casos donde pueden ayudar a mejorar las consultas, pero otros donde no

- estas son estructuras auxiliares y nosotros como usuarios no podemos verlas

# ¿Cuál es el orden de ejecución de una consulta SQL?✔️

Recordemos que sql es un lenguaje declarativo (le decimos qué hacer) y de programación estructurado especializado en almacenamiento y consulta de bases de datos con carácter o estructura relacional

Veamoslo con un ejemplo



Flujo lógico de ejecución

El motor SQL ejecuta esta consulta en este orden:

1. **FROM** → Se determinan las tablas de origen y se hacen los JOINs.  
   (Primero se juntan y generan las filas base).
2. **WHERE** → Se filtran las filas que cumplen la condición.  
   (Elimina las filas que no interesan antes de agrupar).
3. **GROUP BY** → Se agrupan las filas según las columnas especificadas.  
   (Ejemplo: agrupar todas las ventas por ciudad).
4. **HAVING** → Filtra los grupos creados por el GROUP BY.  
   (Ejemplo: mostrar solo ciudades con más de 10,000 en ventas).
5. **SELECT** → Se eligen las columnas o expresiones a mostrar.  
   (Aquí se calculan los agregados como *SUM*, *AVG*, etc.)
6. **DISTINCT** → Se eliminan duplicados (si está presente).
7. **ORDER BY** → Ordena los resultados.  
   (Ejemplo: de mayor a menor venta total).
8. **LIMIT / TOP** → Devuelve solo las primeras N filas (si está presente).

Es decir

 Primero: **de dónde** (FROM).

 Luego: **qué filas valen** (WHERE).

 Después: **cómo agrupar** (GROUP BY).

 Más tarde: **qué grupos valen** (HAVING).

 Luego: **qué mostrar** (SELECT).

 Finalmente: **cómo ordenar y limitar** (ORDER BY, LIMIT).

1. ¿Un profesional debe memorizar el orden de ejecución?

Sí y no:

* **Sí conviene** tenerlo claro porque ayuda a **entender por qué ciertas cosas funcionan o no**, por ejemplo:
  + No puedes usar un alias del SELECT en el WHERE → porque WHERE se ejecuta antes que SELECT.
  + HAVING solo filtra sobre resultados ya agrupados → porque va después del GROUP BY.
* **No necesitas memorizarlo como una lista rígida**: con la práctica y al resolver consultas, se vuelve natural. Lo importante es **entender la lógica** detrás.

2. ¿El orden de ejecución es el mismo en todos los motores SQL?

* **Sí, en lo esencial**. El estándar SQL (ISO/ANSI) define un **orden lógico de ejecución** que todos los motores respetan.
* Ejemplos: MySQL, SQL Server, PostgreSQL, Oracle → todos procesan con la misma secuencia (FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY → LIMIT).
* Puede haber **diferencias internas en la optimización** (*físico* vs *lógico*):
  + El motor puede reordenar operaciones por eficiencia (por ejemplo, empujar un filtro del WHERE antes de hacer un join).
  + Pero el **resultado final siempre respeta el orden lógico** definido por SQL.

3. ¿El motor de SQL tiene perfectamente definido qué declaración ejecuta primero y cuál después?

Exacto ✅

* El estándar SQL define un **“flujo lógico”** que es el que expliqué.
* Internamente, cada motor tiene un **optimizador de consultas** que traduce tu SQL en un **plan de ejecución físico**.
  + Este plan puede cambiar el orden real de operaciones (para mejorar rendimiento).
  + Pero siempre mantiene el mismo resultado como si hubiera seguido el orden lógico.

En conclusión:

* Como profesional, **sí debes conocer el orden lógico**, porque explica limitaciones y comportamientos de SQL.
* Ese orden es **estándar entre motores**.
* Los motores internamente pueden optimizar, pero siempre respetan el resultado lógico.

## Cómo pensar una consulta SQL

1. **FROM** → ¿De dónde saco los datos?
   * Tablas, vistas, joins.
   * “¿Qué fuente voy a usar?”
2. **WHERE** → ¿Qué filas quiero conservar?
   * Filtro fila por fila.
   * “De todo lo que traje, qué sí me sirve.”
3. **GROUP BY** → ¿Necesito agrupar?
   * Por cliente, por producto, por región, etc.
   * “Quiero ver totales por categoría.”
4. **HAVING** → ¿Qué grupos valen?
   * Filtro sobre los grupos, no sobre filas.
   * “Muéstrame solo clientes con más de 10 compras.”
5. **SELECT** → ¿Qué columnas o cálculos mostrar?
   * Campos, agregados (SUM, AVG), alias.
   * “Ahora sí, qué quiero ver en pantalla.”
6. **ORDER BY** → ¿En qué orden?
   * Ascendente, descendente.
   * “Quiero que aparezca de mayor a menor venta.”
7. **LIMIT / TOP** → ¿Cuántos resultados quiero?
   * “Dame solo los 10 primeros.”

Al momento de crear una consulta primero debo plantear el orden lógico y después el orden sintáctico

Las principales clausulas a tener presentes:

| **Orden lógico** | **Cláusula SQL** | **Qué hace** |
| --- | --- | --- |
| 1 | FROM | De dónde se obtienen los datos (tablas, joins, subconsultas) |
| 2 | WHERE | Filtra filas antes de cualquier agrupación |
| 3 | GROUP BY | Agrupa filas según columnas especificadas |
| 4 | HAVING | Filtra los grupos creados por GROUP BY |
| 5 | SELECT | Selecciona columnas o cálculos finales, aplica agregados |
| 6 | DISTINCT | Elimina duplicados si aplica |
| 7 | ORDER BY | Ordena los resultados finales |
| 8 | LIMIT / TOP | Limita el número de resultados |

💡 **Tip de profesional:** no necesitas memorizar de memoria cada vez, sino **tenerlo tan presente que cuando planees una consulta, naturalmente pienses “primero de dónde, luego filtrar, luego agrupar…”**.

**🔵 Avanzado (21–30)❕**

# ¿Qué es una CTE (Common Table Expression) y cuándo usarla?

# ¿Cómo calcularías un promedio móvil con SQL?

# ¿Qué diferencias hay entre EXISTS, IN y NOT IN?

# ¿Cómo manejarías valores NULL en funciones agregadas?

# ¿Qué son las funciones analíticas y cómo se diferencian de las agregadas?

# ¿Cómo detectarías outliers usando SQL?

# ¿Qué es una transacción y cuáles son sus propiedades (ACID)?

# ¿Qué es una vista (view) y cuáles son sus ventajas y desventajas?

# ¿Cómo optimizarías una consulta lenta?

# ¿Cómo harías una consulta para encontrar la segunda fila más alta en una tabla de ventas?

# Enfoque en ciencia de datos

## Flujo de ciencia de datos en SQL

**🔶 Análisis de Datos y Negocio (1–5)**

¿Cómo identificarías a los clientes más valiosos para un negocio usando SQL o Python? ✔️  
(Pista: piensa en RFM – Recency, Frequency, Monetary)

Definiría una consulta con where para filtrar, a partir de un umbral e indicador disponible, a aquellos que más se destaquen en dicho indicador

✅ **Recency (Recencia):** Cuánto tiempo ha pasado desde la última compra → Clientes recientes son más valiosos.  
✅ **Frequency (Frecuencia):** Cuántas veces ha comprado el cliente → Más compras, más valioso.  
✅ **Monetary (Monetario):** Cuánto dinero ha gastado → Más gasto, más valioso.

Es decir, en sql haría una consulta analizando cuidadosamente las relaciones de tablas y utilizaría Python para analizar gráficamente esa consulta.

## Si las ventas de un producto disminuyen, ¿cómo investigarías las causas usando datos? ✔️

Analizaría algún, o preferentemente, algunos indicadores disponibles, priorizando la fecha y encontrar relaciones temporales con el aumento o disminución de estas. Pero también algunos otros indicadores disponibles para ver e interpretar cómo se mueve el valor de la venta en correspondencia con dicho indicador o indicadores correspondientes

Qué son las KPIs ✔️

En **ciencia de datos**, las **KPIs** (*Key Performance Indicators* o **Indicadores Clave de Desempeño**) son métricas cuantitativas que se utilizan para evaluar el rendimiento de modelos, procesos, proyectos o iniciativas de análisis de datos. Su función es **medir el progreso hacia objetivos específicos** y ayudar en la toma de decisiones informadas.

 **KPIs de modelos de machine learning**

* Accuracy, Precision, Recall, F1-score, AUC-ROC, RMSE, MAE, Log-loss.

 **KPIs de impacto en negocio**

* ROI, aumento de ventas, reducción de costos, mejora en conversión, disminución de churn.

 **KPIs del proceso de ciencia de datos**

* Tiempo de desarrollo, tiempo de entrenamiento/predicción, número de iteraciones, éxito en despliegue.

 **KPIs de calidad de datos**

* % de valores faltantes/duplicados, nivel de completitud, frecuencia de actualización, consistencia.

 **KPIs de uso del modelo en producción**

* Tasa de adopción, número de predicciones por día, tiempo de respuesta, tasa de errores.
* Permiten **medir objetivamente el valor** que genera un proyecto de ciencia de datos.
* Ayudan a **alinear los objetivos técnicos con los de negocio**.
* Facilitan la **comunicación con stakeholders no técnicos**.

¿Cómo calcularías la tasa de retención de usuarios mes a mes con SQL?✔️

Agruparía por mes el la suma total de usuarios para observar cómo varia la cantidad de usuarios en función del mes (la gráfica la haría con Python o Power BI). A partir de aquí tomar o seguir investigando acciones

**🔷 Series Temporales y Fechas (6–10)**

¿Cómo agruparías datos por semana, mes o trimestre en SQL? ¿Y en Pandas?

¿Cómo calcularías el crecimiento porcentual mes a mes de una métrica?

¿Qué es un “lag” en series de tiempo y cómo lo aplicarías en una base de datos?

¿Cómo identificarías una tendencia estacional en una serie temporal?

¿Qué harías si te encuentras con timestamps en diferentes zonas horarias?

**🔸 Calidad de Datos y Preprocesamiento (11–13)**

¿Cómo detectarías outliers en una variable numérica usando SQL o Python?

¿Qué pasos seguirías si una tabla tiene muchas columnas con más del 50% de valores nulos?

¿Cómo tratarías datos categóricos con errores tipográficos o formatos inconsistentes (como “Male”, “M”, “male”)?

**🔺 Aplicación Práctica y Pensamiento Analítico (14–15)**

Te entregan un dataset de usuarios y eventos. ¿Cómo calcularías la conversión desde el registro hasta una compra?

Tienes un dataset con 100 columnas. ¿Cómo identificarías las variables más relevantes para predecir una salida (target)?