📘 SQL desde Cero

La guía esencial para empezar en el mundo de los datos

por Ian Saura

✳️ Capítulos y contenido

1. Introducción al mundo de los datos y SQL

- ¿Qué es una base de datos? (Relacional vs. No relacional)
- ¿Qué es SQL y cómo se usa en la industria?
- Casos reales de uso en Data Analyst / Data Engineer / Data Scientist
- Diferencia entre OLTP y OLAP

2. Primer contacto con SQL

- ¿Cómo se estructura una consulta SQL?
- SELECT básico con ejemplos
- FROM y alias de tablas
- LIMIT y ordenamiento (`ORDER BY`)
- Primeros desafíos guiados

3. Filtrado de datos

- `WHERE` con condiciones simples y múltiples (`AND`, `OR`)
- `BETWEEN`, `IN`, `LIKE`, `IS NULL`
- Operadores relacionales
- Prácticas con un dataset tipo “clientes y compras”

---

4. Ordenamiento y lógica de ejecución

- `ORDER BY`, `DESC`, `ASC`
- ¿Cómo se ejecuta internamente una consulta SQL?
- Mini desafío: ordenar productos según precio, luego nombre

---

5. Agrupaciones y funciones agregadas

- `GROUP BY` y `HAVING`
- Funciones: `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`
- Cómo agrupar por múltiples columnas
- Ejercicios con análisis de ventas

---

6. Relaciones y JOINs

- Claves primarias y foráneas
- `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL OUTER JOIN`
- Alias, desambiguación de columnas
- Comparativa visual (diagrama tipo Venn)
- Casos reales: clientes, pedidos, productos

---

7. Subconsultas y CTEs

- Subqueries en `WHERE`, `SELECT`, `FROM`
- `WITH` (Common Table Expressions)

- Ventajas de legibilidad y optimización
- Ejercicio: clientes que gastaron más que el promedio

---

8. Funciones avanzadas

- Funciones de fecha (`CURRENT_DATE`, `DATE_PART`, etc.)
- Funciones condicionales: `CASE WHEN`
- Concatenación, manipulación de strings
- Bonus: funciones específicas por motor (Postgres, MySQL)

---

9. Creación y manipulación de tablas

- `CREATE TABLE`, `INSERT`, `UPDATE`, `DELETE`
- `DROP`, `TRUNCATE`
- Tipos de datos
- Integridad referencial y constraints

---

10. Proyecto práctico final

- Dataset realista (ventas e-commerce, por ejemplo)
- Desafíos crecientes:
  - KPIs de ventas
  - Clientes top
  - Productos sin ventas
  - Segmentos geográficos
- Solución guiada paso a paso + explicación de cada consulta

---

11. Bonus: Cómo practicar en tu compu

- Instalar SQLite o usar DBeaver con Postgres
- Dataset de prueba para descargar

- Plantilla SQL con queries comunes
- Link a playgrounds online

---

12. Apéndices

- Glosario de términos
- Cheatsheet visual
- Preguntas frecuentes (FAQ)
- Tabla resumen de tipos de JOIN
- Checklist de lo que deberías saber al terminar

Capítulo 1 — Introducción al mundo de los datos y SQL

🌍 ¿Qué es una base de datos?

Una base de datos es un sistema organizado que permite **almacenar, acceder y manipular información estructurada**. Es el corazón de prácticamente cualquier aplicación, negocio o plataforma digital.

Existen distintos tipos de bases de datos, pero las más utilizadas en el mundo del análisis y la ingeniería de datos son las **relacionales**.

🧱 ¿Qué es una base de datos relacional?

Una base de datos relacional guarda la información en **tablas**, parecidas a hojas de cálculo, donde:

- Cada **fila** representa un registro (ej.: un cliente, un producto, una venta)
- Cada **columna** representa un atributo (ej.: nombre, fecha, precio)

Estas tablas están **relacionadas entre sí** mediante claves, lo que permite combinar información de distintas fuentes de forma estructurada.

Ejemplo visual:

cliente_id	nombre	ciudad
1	Ana	CABA
2	Bruno	Rosario
3	Camila	Mendoza

💬 ¿Qué es SQL?

SQL (pronunciado “sequel” o “ese-cu-e-le”) significa **Structured Query Language**, y es el lenguaje estándar para **consultar, manipular y administrar bases de datos relacionales**.

No importa qué herramienta estés usando: si hay una base de datos relacional, probablemente puedas hablarle con SQL.

✅ ¿Para qué se usa?

- Consultar datos históricos (ventas, usuarios, métricas)
- Filtrar, ordenar y agrupar información
- Unir múltiples tablas
- Crear nuevas tablas o transformar las existentes
- Automatizar reportes y dashboards
- Soportar decisiones de negocio basadas en datos

👨‍💻 ¿Quiénes usan SQL en su trabajo?

Rol	¿Cómo usa SQL?
Data Analyst	Hace análisis, métricas, dashboards
Data Engineer	Construye pipelines, transforma y modela datos
Data Scientist	Extrae datos limpios para entrenar modelos
Business Analyst	Consulta métricas para tomar decisiones
Product Manager	Hace análisis exploratorio o valida hipótesis

Si trabajás con datos, **SQL es tu superpoder base**.

🧠 ¿Qué diferencia hay entre OLTP y OLAP?

Una distinción fundamental en el mundo de los datos:

- **OLTP (Online Transaction Processing)**
  - Bases de datos operativas, donde ocurren las transacciones del día a día
  - Ejemplo: cuando un usuario hace una compra en una app
  - Necesitan velocidad, consistencia y concurrencia
- **OLAP (Online Analytical Processing)**
  - Bases de datos analíticas, pensadas para hacer análisis históricos
  - Se usan para BI, dashboards, modelos analíticos
  - Optimizadas para lectura masiva y consultas complejas

👉 **Este curso se enfoca en el uso analítico de SQL**, es decir, en cómo usarlo para responder preguntas de negocio y trabajar con datos históricos.

✨ Conclusión

Aprender SQL es como aprender a leer los datos de tu empresa. Es un lenguaje simple, lógico y poderoso.

Y lo mejor: con unas pocas instrucciones podés empezar a hacer magia con millones de filas.

En el próximo capítulo vas a escribir tus primeras consultas SQL.

Capítulo 2 — Primeros pasos con SQL

✍️ ¿Cómo se escribe una consulta en SQL?

Una consulta SQL es una instrucción que **le pide algo a la base de datos**, y esta devuelve una respuesta en forma de tabla.

La estructura más básica de una consulta SQL es:

```
SELECT columna1, columna2
FROM nombre_de_tabla;
```

Esto significa: “Mostrame las columnas 1 y 2 de la tabla que se llama `nombre_de_tabla`.”

🧪 Ejemplo 1: Selección simple

Imaginemos una tabla llamada `clientes` con esta información:

cliente_id	nombre	ciudad
1	Ana	Buenos Aires
2	Luis	Córdoba
3	Sol	Mendoza

Consulta:

```
SELECT nombre, ciudad
FROM clientes;
```

Resultado:

nombre	ciudad
Ana	Buenos Aires
Luis	Córdoba
Sol	Mendoza

📦 SELECT: ¿Qué columnas querés ver?

- `SELECT *` significa “mostrame todas las columnas”
- Podés pedir una o más columnas, separadas por coma
- También podés renombrarlas con `AS`

```
SELECT nombre AS nombre_cliente, ciudad
FROM clientes;
```

📦 FROM: ¿De qué tabla?

El `FROM` indica de qué **tabla o vista** querés leer los datos.

> ⚠️ ¡IMPORTANTE! Si te olvidás el FROM, la consulta no funciona.

🔤 Cómo renombrar columnas y usar alias

Podés usar `AS` para darle un nombre más claro a una columna o a una tabla:

```
SELECT nombre AS cliente, ciudad AS localidad
FROM clientes AS c;
```

Esto es útil para:

- Mejorar la legibilidad del resultado
- Evitar conflictos cuando usás varias tablas

🔢 LIMIT: Limitar la cantidad de filas

Cuando una tabla tiene millones de filas, no querés que te devuelva todo. Usá `LIMIT` para restringir:

```
SELECT *
FROM clientes
LIMIT 10;
```

📊 ORDER BY: Ordenar los resultados

Usás `ORDER BY` para ordenar los datos por una o más columnas.

```
SELECT nombre, ciudad
FROM clientes
ORDER BY nombre;
```

- Por defecto, el orden es ascendente (A-Z, 1-10)
- Para descendente usás `DESC`

```
ORDER BY nombre DESC;
```

🛠️ Orden de ejecución (no es el que vos escribís)

Aunque vos escribís así:

```
SELECT nombre
FROM clientes
WHERE ciudad = 'Córdoba'
ORDER BY nombre;
```

La base de datos lo ejecuta en este orden:

1. `FROM`
2. `WHERE`
3. `SELECT`
4. `ORDER BY`

Esto es clave para entender errores y comportamientos raros. ¡Memorizalo!

⚠️ Errores comunes y cómo evitarlos

Error	Solución
“Column does not exist”	Revisá que el nombre esté bien escrito
“Syntax error near...”	Asegurate de que el punto y coma esté al final `;`

Error	Solución
“Ambiguous column name”	Usá alias si estás usando varias tablas
Todo en mayúscula o minúscula	SQL no distingue mayúsculas en palabras clave, pero sí en strings

📘 Buenas prácticas desde el día 1

✅ Usá alias cortos pero claros

✅ Usá `AS` para que tu output tenga nombres legibles

✅ Usá `LIMIT` cuando estés probando queries

✅ Escribí en mayúsculas las palabras clave: `SELECT`, `FROM`, `ORDER BY`

🔍 Dataset de práctica

Te dejamos un pequeño dataset que vas a usar a lo largo de esta guía. Simula un ecommerce:

Tabla: clientes

cliente_id	nombre	ciudad
1	Laura	Córdoba
2	Marcos	Rosario
3	Lucía	CABA

Tabla: compras

compra_id	cliente_id	monto	fecha
101	1	3200	2023-01-10
102	2	1500	2023-01-12
103	3	1800	2023-01-15

🧠 Desafío guiado

Objetivo: Queremos ver el nombre y ciudad de todos los clientes, ordenados alfabéticamente.

Paso 1: Mostrar las columnas

```
SELECT nombre, ciudad
FROM clientes;
```

Paso 2: Ordenar por nombre

```
SELECT nombre, ciudad
FROM clientes
ORDER BY nombre;
```

Paso 3: Ver sólo los primeros 2

```
SELECT nombre, ciudad
FROM clientes
ORDER BY nombre
LIMIT 2;
```

✅ ¡Felicitaciones! Escribiste tu primera consulta profesional.

⬅️ Próximo capítulo

En el Capítulo 3 vamos a aprender a **filtrar datos con condiciones** usando `WHERE`, `IN`, `LIKE`, `IS NULL` y más.

Es el paso clave para hacer análisis específicos y responder preguntas de negocio.

Vamos con el **Capítulo 3** de tu guía profesional **SQL desde Cero**! Este capítulo está diseñado para profundizar en **filtrado de datos**, una habilidad central para cualquier analista o ingeniero de datos.

Capítulo 3 — Filtrar datos con WHERE y operadores

🔍 ¿Para qué sirve `WHERE`?

La cláusula `WHERE` te permite **filtrar filas** que cumplen ciertas condiciones. En lugar de traer toda la tabla, seleccionás **solo lo que necesitás**.

📦 Sintaxis básica

```
SELECT columnas
FROM tabla
WHERE condición;
```

Ejemplo:

```
SELECT nombre, ciudad
FROM clientes
WHERE ciudad = 'Rosario';
```

Resultado:

nombre	ciudad
Marcos	Rosario

⚙️ Operadores básicos

Operador	Descripción	Ejemplo
=	Igual a	ciudad = 'Córdoba'
<> o !=	Distinto de	ciudad <> 'Córdoba'
>	Mayor que	monto > 1000
<	Menor que	monto < 5000
>=	Mayor o igual	monto >= 2000

Operador	Descripción	Ejemplo
<=	Menor o igual	monto <= 1500

📂 Filtrar por múltiples condiciones

Usás `AND` y `OR` para combinar condiciones:

```
SELECT *
FROM compras
WHERE monto > 1000 AND fecha >= '2023-01-12';
```

O con `OR`:

```
SELECT *
FROM compras
WHERE ciudad = 'CABA' OR ciudad = 'Rosario';
```

Tip: Si combinás `AND` y `OR`, usá paréntesis:

```
WHERE ciudad = 'CABA' AND (monto > 1000 OR fecha < '2023-01-15')
```

✅ El operador `IN`

Sirve para chequear si un valor está en una lista.

```
SELECT *
FROM clientes
WHERE ciudad IN ('CABA', 'Rosario');
```

Esto es lo mismo que usar varios `OR`, pero más limpio.

🔎 Buscar patrones con `LIKE`

El operador `LIKE` permite buscar textos parcialmente:

Símbolo	Significado
`%`	Cualquier secuencia
`_`	Un solo carácter

Ejemplos:

```
SELECT *
FROM clientes
WHERE nombre LIKE 'L%';  -- empieza con L
```

```
WHERE ciudad LIKE '%ba%'  -- contiene "ba"
```

🔘 Valores nulos: `IS NULL` y `IS NOT NULL`

Las bases de datos pueden tener valores faltantes (`NULL`). Para trabajar con ellos:

```
SELECT *
FROM clientes
WHERE ciudad IS NULL;
```

O para excluirlos:

```
WHERE ciudad IS NOT NULL;
```

> ⚠️ Nunca uses = NULL. Siempre IS NULL.

🔄 BONUS: El operador `BETWEEN`

Sirve para rangos:

```
SELECT *
FROM compras
```

```
WHERE fecha BETWEEN '2023-01-10' AND '2023-01-15';
```

Incluye ambos extremos del rango.

🧠 Desafíos guiados

Desafío 1

Obtené todos los clientes que NO son de CABA:

```
SELECT *
FROM clientes
WHERE ciudad <> 'CABA';
```

Desafío 2

Mostrá las compras cuyo monto fue mayor o igual a $2000:

```
SELECT *
FROM compras
WHERE monto >= 2000;
```

Desafío 3

Mostrá clientes que vivan en Córdoba o Mendoza:

```
SELECT *
FROM clientes
WHERE ciudad IN ('Córdoba', 'Mendoza');
```

🧘 Resumen de lo aprendido

- `WHERE` filtra filas según condiciones.
- Combinás condiciones con `AND`, `OR`, paréntesis.

- `IN`, `LIKE`, `BETWEEN`, `IS NULL` amplían tu poder de filtrado.
- Esta es la base para análisis real.

📦 Qué sigue

En el **Capítulo 4** vas a aprender cómo **crear columnas nuevas con expresiones y funciones SQL**, como operaciones matemáticas, manipulación de texto, fechas y más.

Ideal para enriquecer tus resultados y hacer análisis dinámico.

Vamos con el **Capítulo 4** de tu guía premium "SQL desde Cero"! En este capítulo, vas a aprender a **crear columnas calculadas y usar funciones**, un paso clave para pasar de mostrar datos a **analizarlos activamente**.

Capítulo 4 — Crear columnas nuevas con funciones y expresiones

---

📐 ¿Qué es una expresión en SQL?

Una **expresión** es una operación que se hace sobre una o más columnas para generar un nuevo valor. Esto puede ser:

- Una cuenta matemática: `precio * cantidad`
- Un texto modificado: `UPPER(nombre)`
- Una lógica condicional: `CASE WHEN ... THEN ...`

> Todo lo que escribas en el SELECT puede ser una columna nueva calculada.

🔢 Operaciones matemáticas

✅ Operadores comunes

Operador	Función	Ejemplo
`+`	Suma	`precio + 100`
`-`	Resta	`precio - descuento`
`*`	Multiplicación	`precio * cantidad`
`/`	División	`precio / 2`

🧪 Ejemplo

```
SELECT compra_id, monto, monto * 1.21 AS monto_con_iva
FROM compras;
```

🧠 Funciones más comunes

Las **funciones** son comandos predefinidos que transforman los datos. Se aplican dentro del `SELECT`.

🧮 Funciones matemáticas

```
ABS(x)      -- Valor absoluto
ROUND(x, n)  -- Redondear con n decimales
CEIL(x)     -- Redondear hacia arriba
FLOOR(x)    -- Redondear hacia abajo
```

Ejemplo:

```
SELECT monto, ROUND(monto * 1.21, 2) AS total_redondeado
FROM compras;
```

🔤 Funciones de texto

```
UPPER(texto)      -- Todo en mayúsculas
LOWER(texto)      -- Todo en minúsculas
LENGTH(texto)     -- Largo del texto
```

```
SUBSTRING(texto FROM inicio FOR largo) -- Cortar texto
TRIM(texto)        -- Quitar espacios
CONCAT(a, b)       -- Unir textos
```

Ejemplo:

```
SELECT nombre, UPPER(nombre) AS nombre_mayuscula
FROM clientes;
```

📆 Funciones de fecha

```
CURRENT_DATE           -- Fecha de hoy
EXTRACT(YEAR FROM fecha) -- Año de una fecha
DATE_PART('month', fecha) -- Mes
AGE(fecha)             -- Tiempo desde esa fecha
```

Ejemplo:

```
SELECT fecha, EXTRACT(YEAR FROM fecha) AS año
FROM compras;
```

🔄 Condiciones con `CASE WHEN`

Una de las herramientas más poderosas en SQL.

```
SELECT
  monto,
  CASE
    WHEN monto >= 2000 THEN 'Alta'
    WHEN monto >= 1000 THEN 'Media'
    ELSE 'Baja'
  END AS categoria
FROM compras;
```

Esto crea una nueva columna categorizando montos.

> 🧠 Pensá CASE como un IF o SI en Excel.

🚨 Buenas prácticas

- Siempre **renombrá** las columnas calculadas con `AS`
- Evitá operaciones dentro de `WHERE`, hacelas antes si podés
- Redondeá montos con `ROUND` para evitar muchos decimales
- Usá funciones de fecha para crear análisis temporales

🎯 Desafío guiado

Queremos crear una consulta que:

- Muestre el `nombre`, `ciudad` y una columna `origen` que diga:
  - "Interior" si la ciudad no es CABA
  - "Capital" si es CABA

```
SELECT
  nombre,
  ciudad,
  CASE
    WHEN ciudad = 'CABA' THEN 'Capital'
    ELSE 'Interior'
  END AS origen
FROM clientes;
```

💡 BONUS: Combiná funciones

```
SELECT
  cliente_id,
```

```
  CONCAT(UPPER(nombre), ' de ', ciudad) AS presentacion
FROM clientes;
```

Esto devuelve cosas como:

"LAURA de Córdoba"

🔚 ¿Qué aprendiste?

- Cómo crear columnas nuevas con operaciones y funciones
- Cómo transformar texto, fechas y números
- Cómo usar lógica condicional con `CASE`
- Cómo combinar funciones para análisis más potentes

⏭️ Próximo capítulo

En el **Capítulo 5**, vas a aprender sobre **agregaciones**: cómo usar `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()` y cómo agrupar resultados con `GROUP BY`.

Esto es esencial para pasar de filas individuales a **resúmenes por grupo**.

Capítulo 5 — Agregaciones y Resúmenes con GROUP BY

📊 ¿Qué son las funciones de agregación?

Las funciones de agregación permiten **resumir** datos en lugar de ver fila por fila. Son esenciales para todo análisis de datos.

📦 Funciones de agregación principales

Función	Qué hace	Ejemplo
`COUNT()`	Cuenta la cantidad de filas	`COUNT(*)`
`SUM()`	Suma los valores numéricos	`SUM(monto)`

Función	Qué hace	Ejemplo
`AVG()`	Calcula el promedio	`AVG(monto)`
`MIN()`	Devuelve el valor mínimo	`MIN(fecha)`
`MAX()`	Devuelve el valor máximo	`MAX(fecha)`

✅ Ejemplo básico

```sql
SELECT COUNT(*) AS total_clientes
FROM clientes;
```

```sql
SELECT SUM(monto) AS total_ventas
FROM compras;
```

📂 Agrupar con `GROUP BY`

Si querés ver **totales por ciudad**, por ejemplo:

```sql
SELECT ciudad, COUNT(*) AS cantidad_clientes
FROM clientes
GROUP BY ciudad;
```

Resultado:

ciudad	cantidad_clientes
CABA	45
Córdoba	22
Rosario	18

🧠 Combinar funciones agregadas

Podés usar más de una agregación a la vez:

```sql
SELECT ciudad,
       COUNT(*) AS clientes,
       AVG(edad) AS edad_promedio
FROM clientes
GROUP BY ciudad;
```

📌 Filtrar luego del agrupamiento: `HAVING`

Cuando usás `GROUP BY`, no podés filtrar con `WHERE` sobre las funciones agregadas. Para eso está `HAVING`.

```sql
SELECT ciudad, COUNT(*) AS clientes
FROM clientes
GROUP BY ciudad
HAVING COUNT(*) > 20;
```

🚨 Diferencia entre `WHERE` y `HAVING`

Cláusula	Cuándo se evalúa	Para qué sirve
`WHERE`	Antes del agrupamiento	Filtrar filas individuales
`HAVING`	Después del GROUP BY	Filtrar grupos agregados

🔍 Ejemplo completo

Mostrar el total de ventas por ciudad donde se hayan hecho más de 10 compras:

```
sql

SELECT ciudad, SUM(monto) AS total_ventas
FROM compras
GROUP BY ciudad
HAVING COUNT(*) > 10;
```

🧩 BONUS: Agrupaciones por fechas

```
sql

SELECT
  DATE_PART('month', fecha) AS mes,
  COUNT(*) AS compras_mes
FROM compras
GROUP BY DATE_PART('month', fecha);
```

💡 Buenas prácticas

- Siempre renombrá las columnas agregadas con `AS`
- Usá `HAVING` para filtrar agregados

- Podés combinar `GROUP BY` con funciones como `CASE` para crear análisis más complejos

---

🧠 Desafío guiado

Queremos conocer:

- Cuántos clientes hay por ciudad
- Sólo ciudades con más de 5 clientes

```
sql

SELECT ciudad, COUNT(*) AS cantidad
FROM clientes
GROUP BY ciudad
HAVING COUNT(*) > 5;
```

🔚 ¿Qué aprendiste?

- Las funciones `COUNT()`, `SUM()`, `AVG()`, `MIN()`, `MAX()`
- Cómo usar `GROUP BY` para crear resúmenes
- Cómo aplicar filtros a esos resúmenes con `HAVING`
- Cómo agrupar por fechas u otras columnas derivadas

Capítulo 6 — JOINs: Cómo unir tablas en SQL

🤝 ¿Qué es un JOIN?

En SQL, un **JOIN** permite **combinar datos de dos o más tablas** en una sola consulta, usando una relación lógica entre ellas.

Por ejemplo: unir la tabla de `clientes` con la de `compras` usando el `cliente_id`.

📌 Tipos de JOIN más comunes

JOIN	Qué hace
`INNER JOIN`	Devuelve solo coincidencias en ambas tablas
`LEFT JOIN`	Devuelve todos los registros de la tabla izquierda + coincidencias
`RIGHT JOIN`	Devuelve todos los registros de la tabla derecha + coincidencias
`FULL JOIN`	Devuelve todos los registros, coincidan o no

> En la práctica, INNER y LEFT cubren el 95% de los casos.

🛠️ Sintaxis básica de un JOIN

```sql
SELECT *
FROM clientes
JOIN compras ON clientes.cliente_id = compras.cliente_id;
```

✅ Ejemplo real

Supongamos:

Tabla `clientes`

cliente_id	nombre
1	Laura
2	Marcos

Tabla `compras`

compra_id	cliente_id	monto
10	1	200
11	1	300

Resultado del JOIN:

```sql
SELECT
  clientes.nombre,
  compras.monto
FROM clientes
JOIN compras ON clientes.cliente_id = compras.cliente_id;
```

nombre	monto
Laura	200
Laura	300

🔎 INNER JOIN: Solo coincidencias

```sql
SELECT ...
FROM clientes
INNER JOIN compras
ON clientes.cliente_id = compras.cliente_id;
```

> No muestra clientes sin compras.

👈 LEFT JOIN: Todo de la izquierda

```
sql

SELECT ...
FROM clientes
LEFT JOIN compras
ON clientes.cliente_id = compras.cliente_id;
```

> Muestra todos los clientes, incluso si no hicieron compras.
>
> En esos casos, los campos de `compras` aparecerán como `NULL`.

🧠 ¿Cuándo usar cada uno?

- `INNER JOIN`: Cuando **solo** te interesa la info con coincidencias.
- `LEFT JOIN`: Cuando querés conservar **todos los registros de la tabla principal** (usualmente clientes, productos, empleados...).

🧩 BONUS: Usar múltiples JOINs

```
sql

SELECT
  c.nombre,
  p.producto,
  f.fecha
FROM facturas f
JOIN clientes c ON f.cliente_id = c.cliente_id
JOIN productos p ON f.producto_id = p.producto_id;
```

🎯 Desafío guiado

Mostrar los nombres de los clientes y sus compras (si las tienen):

```
sql

SELECT
  c.nombre,
  cp.monto
FROM clientes c
LEFT JOIN compras cp
ON c.cliente_id = cp.cliente_id;
```

🚨 Buenas prácticas

- Usá **alias** para tablas (`c`, `cp`, etc.) para legibilidad
- Asegurate de **usar la clave correcta** al unir tablas
- Siempre verificá que el resultado tenga sentido: no se duplique o pierda filas sin explicación
- Si ves muchos NULLs, analizá si estás usando correctamente LEFT o INNER

🔚 ¿Qué aprendiste?

- Cómo conectar dos tablas con JOINs
- Diferencias entre INNER y LEFT JOIN
- Cómo escribir consultas multi-join para dashboards y reportes

Capítulo 7 — Subconsultas y CTEs (`WITH`) en SQL

🧠 ¿Por qué aprender esto?

Cuando tus consultas empiezan a complicarse, necesitas una forma de **estructurar mejor el código**, **evitar repeticiones** y **hacerlo más legible**.

Para eso existen:

- **Subconsultas**: consultas dentro de otras consultas
- **CTEs (Common Table Expressions)**: consultas temporales que se pueden reutilizar

---

🔍 ¿Qué es una subconsulta?

Una **subconsulta** es una consulta SQL que se **incrusta dentro de otra**.

```
sql

SELECT nombre
FROM clientes
WHERE cliente_id IN (
  SELECT cliente_id
  FROM compras
  WHERE monto > 1000
);
```

> Traducción: dame los nombres de los clientes que hicieron compras mayores a 1000.

🔁 Subconsulta como campo

```
sql

SELECT
  nombre,
```

```
  (SELECT COUNT(*)
   FROM compras
   WHERE compras.cliente_id = clientes.cliente_id)
   AS total_compras
FROM clientes;
```

> En cada fila, hace un conteo de compras específicas para ese cliente.

🔁 Subconsulta como tabla

```
sql

SELECT *
FROM (
  SELECT cliente_id, COUNT(*) AS compras
  FROM compras
  GROUP BY cliente_id
) AS resumen
WHERE compras > 5;
```

> Esto es más potente pero más difícil de leer. Por eso existe la siguiente mejora...

🧽 ¿Qué es un CTE (`WITH`)?

Un **CTE** es como **una tabla temporal con nombre**, útil para dividir el problema en partes.

```sql
WITH resumen_compras AS (
  SELECT cliente_id, COUNT(*) AS compras
  FROM compras
  GROUP BY cliente_id
)

SELECT c.nombre, r.compras
FROM resumen_compras r
JOIN clientes c ON r.cliente_id = c.cliente_id
WHERE r.compras > 5;
```

> Mucho más limpio y profesional. Es el estándar en proyectos reales.

🤩 ¿Por qué usar CTEs?

- Reutilizás resultados parciales
- Ganás **legibilidad** al dividir pasos
- Podés tener **múltiples CTEs encadenados**
- Ideal para consultas largas en dashboards

🧠 CTEs encadenados

```sql
WITH resumen_compras AS (
  SELECT cliente_id, COUNT(*) AS compras
  FROM compras
  GROUP BY cliente_id
```

```
),
clientes_filtrados AS (
  SELECT *
  FROM resumen_compras
  WHERE compras > 5
)

SELECT *
FROM clientes_filtrados;
```

🧩 BONUS: Subconsulta vs CTE

Característica	Subconsulta	CTE
Legibilidad	Difícil en consultas largas	Mucho más clara
Reutilizable	No	Sí, podés referenciarla varias veces
Soportada en	Todas las bases	Todas, aunque varía su eficiencia
Recomendación	Evitá anidar muchas	Preferí usar CTEs

🧪 Desafío práctico

Queremos saber el nombre de los clientes que hayan hecho **más de 5 compras**:

```
sql

WITH resumen AS (
  SELECT cliente_id, COUNT(*) AS total
  FROM compras
  GROUP BY cliente_id
)

SELECT c.nombre, r.total
FROM resumen r
JOIN clientes c ON r.cliente_id = c.cliente_id
```

```
WHERE r.total > 5;
```

🔚 ¿Qué aprendiste?

- Qué son subconsultas y cómo usarlas en `WHERE`, `SELECT`, y `FROM`
- Cómo mejorar tus consultas con CTEs usando `WITH`
- Por qué los CTEs son la mejor práctica para consultas profesionales

⏭️ Próximo capítulo

En el **Capítulo 8** cerramos con un bonus: **Cómo practicar SQL localmente con SQLite**

Ideal para practicar sin depender de bases online, ¡directamente en tu compu!

Capítulo 8 — BONUS: Practicá SQL localmente con SQLite

🧰 ¿Por qué usar SQLite?

SQLite es una base de datos **liviana, gratuita y sin necesidad de instalación de servidores**.

Ideal para practicar SQL directamente desde tu computadora, sin depender de bases remotas.

💡 Ventajas de SQLite

- Funciona sin conexión a internet
- No requiere instalación complicada
- Compatible con archivos `.db` que podés guardar, compartir y versionar
- Ideal para probar consultas, hacer pruebas o aprender

📥 ¿Cómo empezar?

Paso 1 – Instalar DB Browser for SQLite

> 🔗 https://sqlitebrowser.org/dl/

Este programa te permite:

- Crear bases desde cero
- Importar archivos CSV
- Escribir consultas SQL con autocompletado
- Ver resultados en una interfaz gráfica amigable

Paso 2 – Crear una base de datos nueva

1. Abrí DB Browser
2. Click en **“New Database”**
3. Guardala como `practica.db`
4. Creá tu primera tabla con el botón “Create Table”

Paso 3 – Crear una tabla desde SQL

```
sql

CREATE TABLE clientes (
  cliente_id INTEGER PRIMARY KEY,
  nombre TEXT,
  email TEXT
);
```

Paso 4 – Insertar datos

```sql
INSERT INTO clientes (nombre, email)
VALUES ('Laura', 'laura@email.com'),
       ('Marcos', 'marcos@email.com');
```

Paso 5 – Consultar la tabla

```sql
SELECT * FROM clientes;
```

✅ ¡Y ya estás practicando SQL en tu propia PC!

📁 BONUS: Usar CSVs para importar datos

¿Querés practicar con más volumen de datos?

1. Exportá cualquier archivo `.csv` desde Excel o Google Sheets
2. En DB Browser, elegí **File > Import > Table from CSV**
3. Dale nombre y confirmá

> Ya tenés una tabla real con datos reales para probar GROUP BY, JOIN, filtros, etc.

🧠 Extra: usar SQLite desde Python

Si más adelante aprendés Python, podés conectar SQLite así:

```python
```

```
import sqlite3
conn = sqlite3.connect('practica.db')
cursor = conn.cursor()
cursor.execute("SELECT * FROM clientes")
print(cursor.fetchall())
```

🧭 ¿Qué aprendiste en este capítulo?

- Cómo instalar SQLite y DB Browser
- Cómo crear, llenar y consultar tablas locales
- Cómo importar CSVs
- Cómo integrarlo con otros lenguajes

🎓 Cierre del curso

¡Felicitaciones! Completaste esta guía práctica **"SQL desde Cero"**.

Aprendiste:

- Las bases sólidas de SQL (SELECT, filtros, agrupaciones)
- Cómo hacer JOINs y estructurar consultas avanzadas
- Cómo practicar sin depender de herramientas externas

🚀 ¿Y ahora qué?

Si querés llevar tu SQL al siguiente nivel, con proyectos reales, consultas complejas y revisión con feedback...
Ya estas listo para 👉 Unirte al
Bootcamp de Data Engineering Inicial-Intermedio

📆 Clases en vivo – Recursos descargables – Comunidad privada

🔗 iansaura.com