NIVEL · DESDE CERO

SQL *desde cero*

SELECT · WHERE · JOIN

las queries que te sacan del 80% de los apuros.

este cheatsheet asume que no sabés SQL — o que lo sabés de memoria pero nunca entendiste del todo *por qué* funciona. cada sección tiene la sintaxis, un ejemplo con datos reales y las variantes que más se usan. nada de teoría innecesaria: solo lo que necesitás para escribir queries útiles desde el primer día. 💌

clientes

id	int · pk
nombre	text
email	text
pais	text
telefono	text · null
activo	bool

pedidos

id	int · pk
cliente_id	int · fk
producto_id	int · fk
fecha	date
cantidad	int
total	numeric

productos

id	int · pk
nombre	text
categoria	text
precio	numeric
stock	int
ventas	int

01 la idea básica — *cómo piensa SQL*

SQL es un lenguaje para hacerle preguntas a una base de datos. cada query sigue la misma lógica: **de dónde** saco los datos (`FROM`), **qué condición** deben cumplir (`WHERE`) y **qué columnas** quiero ver (`SELECT`). el motor siempre procesa en este orden interno:

1	`FROM / JOIN`	junta las tablas necesarias
2	`WHERE`	filtra las filas que no cumplen la condición
3	`GROUP BY`	agrupa las filas por una o más columnas
4	`HAVING`	filtra los grupos (como WHERE pero sobre grupos)
5	`SELECT`	elige las columnas y calcula expresiones
6	`ORDER BY`	ordena el resultado
7	`LIMIT / TOP`	recorta la cantidad de filas devueltas

ojo **ojo —** aunque escribís `SELECT` primero, SQL lo evalúa casi al final. por eso *no podés* usar un alias definido en `SELECT` dentro del `WHERE` de la misma query.

02 SELECT — *elegir qué columnas ver*

SINTAXIS BÁSICA

```
SELECT columna1, columna2 FROM tabla;
SELECT nombre, email FROM clientes;
```

devuelve solo `nombre` y `email` de todos los clientes.

SELECCIONAR TODO

```
SELECT * FROM clientes;
```

el `*` trae todas las columnas. útil para explorar — evitalo en producción.

ALIAS — RENOMBRAR EN EL RESULTADO

```
SELECT nombre AS cliente,
       email AS correo
FROM clientes;
```

`AS` es opcional pero hace el código más legible. los alias no cambian la tabla original.

COLUMNAS CALCULADAS

```
SELECT nombre, precio * cantidad AS total
FROM pedidos;
```

podés operar matemáticamente sobre columnas directamente en el `SELECT` .

DISTINCT — SIN DUPLICADOS

```
SELECT DISTINCT pais FROM clientes;
```

devuelve cada país una sola vez, sin repeticiones.

tip **tip —** usá `DISTINCT` con cuidado en tablas grandes: obliga a comparar todas las filas y puede ser lento. si querés contar únicos, `COUNT(DISTINCT col)` suele ser mejor.

03 WHERE — *filtrar filas*

`WHERE` reduce el conjunto de filas *antes* de que `SELECT` haga su trabajo. solo pasan las filas donde la condición es verdadera.

OPERADORES DE COMPARACIÓN

OP.	SIGNIFICADO	EJEMPLO
`=`	igual	`WHERE pais = 'Argentina'`
`<>` · `!=`	distinto	`WHERE estado <> 'cancelado'`
`>`	mayor que	`WHERE precio > 100`
`<`	menor que	`WHERE edad < 30`
`>=`	mayor o igual	`WHERE stock >= 1`
`<=`	menor o igual	`WHERE descuento <= 0.1`

AND · OR · NOT — COMBINAR

```sql
SELECT * FROM clientes
WHERE pais = 'Mexico' AND activo = 1;
```

```sql
SELECT * FROM clientes
WHERE pais = 'Chile' OR pais = 'Peru';
```

```sql
SELECT * FROM productos
WHERE NOT categoria = 'ropa';
```

ojo **ojo —** con `AND` y `OR` mezclados, usá paréntesis para dejar clara la prioridad: `WHERE (a OR b) AND c`.

IN — LISTA DE VALORES

```sql
SELECT * FROM clientes
WHERE pais IN ('Argentina', 'Chile', 'Uruguay');
```

equivale a una cadena de `OR`, pero más limpio.

BETWEEN — RANGO INCLUSIVO

```sql
SELECT * FROM pedidos
WHERE total BETWEEN 100 AND 500;
```

```sql
SELECT * FROM pedidos
WHERE fecha BETWEEN '2024-01-01'
                AND '2024-12-31';
```

incluye los extremos. funciona con números, fechas y texto.

LIKE — BUSCAR PATRONES

```sql
SELECT * FROM clientes
WHERE nombre LIKE 'Mar%';  -- Maria, Martin…
WHERE email  LIKE '%@gmail.com';
WHERE codigo LIKE 'A_01';  -- A101, AB01…
```

`%` = cero o más caracteres · `_` = exactamente uno.

IS NULL · IS NOT NULL — VALORES VACÍOS

```sql
SELECT * FROM clientes
WHERE telefono IS NULL;
```

```sql
SELECT * FROM clientes
WHERE telefono IS NOT NULL;
```

ojo **ojo —** nunca uses `= NULL`, no funciona. `NULL` no es cero ni cadena vacía: es *ausencia de dato*. Por eso `NULL = NULL` devuelve falso en SQL.

04 ORDER BY · LIMIT — *ordenar y recortar*

ORDER BY — ORDENAR RESULTADOS

```sql
SELECT nombre, edad FROM clientes
ORDER BY edad;                -- ASC por defecto

SELECT nombre, total FROM pedidos
ORDER BY total DESC;          -- mayor a menor

SELECT * FROM clientes
ORDER BY pais ASC, nombre ASC;
```

primero ordena por país; dentro de cada país, por nombre.

LIMIT · TOP — LIMITAR FILAS

```sql
-- MySQL · PostgreSQL · SQLite
SELECT * FROM productos
ORDER BY precio DESC LIMIT 10;

-- SQL Server
SELECT TOP 10 * FROM productos
ORDER BY precio DESC;
```

```sql
SELECT * FROM productos
ORDER BY precio DESC
LIMIT 10 OFFSET 20;   -- paginación
```

`OFFSET 20` se salta las primeras 20 filas. útil para paginar.

05 funciones de agregación — *resumir datos*

las funciones de agregación **colapsan** múltiples filas en un solo valor. se usan con o sin `GROUP BY`.

FUNCIÓN	QUÉ HACE	EJEMPLO
`COUNT(*)`	cuenta todas las filas	`SELECT COUNT(*) FROM pedidos`
`COUNT(col)`	cuenta filas donde `col` no es NULL	`SELECT COUNT(email) FROM clientes`
`SUM(col)`	suma los valores de una columna	`SELECT SUM(total) FROM pedidos`
`AVG(col)`	calcula el promedio	`SELECT AVG(precio) FROM productos`
`MIN(col)`	valor más bajo	`SELECT MIN(fecha) FROM pedidos`
`MAX(col)`	valor más alto	`SELECT MAX(total) FROM pedidos`

EJEMPLOS PRÁCTICOS

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

```sql
SELECT
  SUM(total) AS ventas_totales,
  AVG(total) AS ticket_promedio
FROM pedidos;
```

MIN + MAX DE UN TIRÓN

```sql
SELECT
  MAX(total) AS pedido_mas_caro,
  MIN(total) AS pedido_mas_barato
FROM pedidos;
```

ojo **ojo —** no podés mezclar columnas normales con funciones de agregación sin `GROUP BY`. `SELECT nombre, COUNT(*) FROM clientes` falla.

06 GROUP BY · HAVING — *agrupar y filtrar grupos*

`GROUP BY` agrupa las filas por una o más columnas y aplica las funciones de agregación **a cada grupo por separado**.

GROUP BY BÁSICO

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

cuenta cuántos clientes hay en cada país.

```sql
SELECT categoria,
       SUM(ventas) AS total_ventas
FROM productos
GROUP BY categoria
ORDER BY total_ventas DESC;
```

MÚLTIPLES COLUMNAS

```sql
SELECT pais, categoria,
       SUM(total) AS ventas
FROM pedidos
GROUP BY pais, categoria;
```

crea un grupo por cada *combinación única* de país + categoría.

HAVING — FILTRAR GRUPOS

```sql
SELECT pais, COUNT(*) AS clientes
FROM clientes
GROUP BY pais
HAVING COUNT(*) > 100;
```

solo países con más de 100 clientes.

tip **regla de oro —** `WHERE` filtra filas *antes* de agrupar. `HAVING` filtra grupos *después* de agrupar. si no agrupás, no uses HAVING.

07 JOINs — *unir tablas*

un `JOIN` combina filas de dos tablas usando una columna en común. es la operación más útil y más temida de SQL. una vez que la entendés, la mayoría de las queries complejas se vuelven simples.

INNER JOIN — SOLO COINCIDENCIAS

```sql
SELECT c.nombre, p.fecha, p.total
FROM clientes c
INNER JOIN pedidos p ON c.id = p.cliente_id;
```

muestra solo los clientes que **sí** tienen al menos un pedido. los clientes sin pedidos no aparecen.

LEFT JOIN — TODO LO DE LA IZQUIERDA

```sql
SELECT c.nombre, p.total
FROM clientes c
LEFT JOIN pedidos p ON c.id = p.cliente_id;
```

todos los clientes aparecen. los que no tienen pedidos muestran `NULL` en las columnas de pedidos.

PATRÓN CLÁSICO — CLIENTES SIN PEDIDOS (ANTI-JOIN)

```sql
SELECT c.nombre
FROM clientes c
LEFT JOIN pedidos p ON c.id = p.cliente_id
WHERE p.id IS NULL;   -- los que nunca compraron 💔
```

07 JOINs — *el universo completo*

RIGHT JOIN — TODO LO DE LA DERECHA

```
SELECT c.nombre, p.total
FROM clientes c
RIGHT JOIN pedidos p ON c.id = p.cliente_id;
```

todos los pedidos aparecen. en la práctica es poco común: se suele reescribir como `LEFT JOIN` cambiando el orden de las tablas.

FULL OUTER JOIN — TODO DE AMBAS

```
SELECT c.nombre, p.total
FROM clientes c
FULL OUTER JOIN pedidos p
  ON c.id = p.cliente_id;
```

clientes sin pedidos *Y* pedidos sin cliente válido. útil para detectar inconsistencias.

> *ojo* **ojo —** MySQL no soporta `FULL OUTER JOIN` directo. se simula con `LEFT JOIN UNION RIGHT JOIN` .

UNIR MÁS DE DOS TABLAS

```
SELECT c.nombre, p.fecha, pr.nombre AS producto, p.cantidad
FROM pedidos p
INNER JOIN clientes  c  ON p.cliente_id  = c.id
INNER JOIN productos pr ON p.producto_id = pr.id;
```

cada `JOIN` adicional agrega otra tabla. podés encadenar tantos como necesites.

RESUMEN VISUAL — QUÉ TRAE CADA JOIN

TIPO	DEVUELVE	CUÁNDO USARLO
`INNER`	solo filas con coincidencia en ambas tablas	**lo que tienen en común · default seguro**
`LEFT`	todo A + coincidencias de B	**todo A, lo de B que exista**
`RIGHT`	todo B + coincidencias de A	**todo B, lo de A que exista**
`FULL OUTER`	todo de ambas tablas	**detectar filas huérfanas**

DIAGRAMAS RÁPIDOS

INNER JOIN

LEFT JOIN

FULL OUTER

> *tip* **tip —** cuando dudes, empezá con `INNER JOIN` . si te faltan filas, cambiá a `LEFT JOIN` . el 90% del tiempo vas a usar uno de estos dos.

08 subqueries — *queries dentro de queries*

una **subquery** es un `SELECT` dentro de otro `SELECT`. útil cuando el resultado de una query es la condición de otra.

EN EL WHERE

```sql
SELECT nombre, total FROM pedidos
WHERE total > (
  SELECT AVG(total) FROM pedidos
);
```

pedidos cuyo total supera el promedio general. la subquery devuelve un único número.

CON IN

```sql
SELECT nombre FROM clientes
WHERE id IN (
  SELECT cliente_id FROM pedidos
  WHERE total > 1000
);
```

clientes que tienen al menos un pedido mayor a 1000.

EN EL FROM — TABLA DERIVADA

```sql
SELECT pais, promedio
FROM (
  SELECT pais, AVG(total) AS promedio
  FROM pedidos GROUP BY pais
) AS resumen
WHERE promedio > 500;
```

primero calcula el promedio por país (subquery), después filtra esa tabla temporal.

09 CTEs — *named subqueries con WITH*

un **CTE** (Common Table Expression) es como darle un *nombre* a una subquery para usarla después. hace el código muchísimo más legible.

```sql
WITH pedidos_grandes AS (
  SELECT cliente_id, SUM(total) AS gasto
  FROM pedidos
  GROUP BY cliente_id
  HAVING SUM(total) > 5000
)
SELECT c.nombre, pg.gasto
FROM clientes c
INNER JOIN pedidos_grandes pg ON c.id = pg.cliente_id;
```

`pedidos_grandes` es una tabla temporal que existe solo durante esta query. después se usa como cualquier tabla en el `JOIN`.

CTES ENCADENADOS

```sql
WITH
  ventas_mes AS (
    SELECT mes, SUM(total) AS ventas
    FROM pedidos GROUP BY mes
  ),
  promedio_anual AS (
    SELECT AVG(ventas) AS promedio FROM ventas_mes
  )
SELECT mes, ventas
FROM ventas_mes, promedio_anual
WHERE ventas > promedio_anual.promedio;
```

tip **tip** — usá CTEs cuando tengas subqueries anidadas de más de dos niveles. dividen el problema en pasos legibles. 🎀

10 funciones útiles — *para el día a día*

TEXTO

FUNCIÓN	EJEMPLO
UPPER(col)	UPPER(nombre) → 'MARIA'
LOWER(col)	LOWER(email)
LENGTH(col)	LENGTH(codigo) = 5
TRIM(col)	TRIM(' hola ') → 'hola'
CONCAT(a,b)	CONCAT(nombre,' ',ap)
SUBSTRING	SUBSTRING(codigo,1,3)
REPLACE	REPLACE(tel,'-','')

FECHAS · LAS MÁS UNIVERSALES

FUNCIÓN	EJEMPLO
NOW() · GETDATE()	fecha y hora actuales
CURRENT_DATE	solo fecha de hoy
YEAR(col)	YEAR(fecha) = 2024
MONTH(col)	MONTH(fecha) · 1–12
DAY(col)	DAY(fecha) = 1
DATEDIFF(a,b)	días entre dos fechas
DATE_TRUNC	DATE_TRUNC('month',f)

CONDICIONALES — EL IF/ELSE DE SQL

```sql
SELECT nombre,
  CASE
    WHEN total > 1000 THEN 'VIP'
    WHEN total >  500 THEN 'Frecuente'
    ELSE 'Ocasional'
  END AS segmento
FROM clientes;
```

`CASE WHEN` crea una columna nueva basada en condiciones.

```sql
SELECT nombre,
  COALESCE(telefono, email, 'sin contacto')
    AS contacto
FROM clientes;
```

`COALESCE` devuelve el primer valor **no-NULL** de la lista. imprescindible para datos incompletos.

```sql
SELECT nombre,
  NULLIF(descuento, 0) AS descuento_real
FROM productos;
```

`NULLIF` devuelve `NULL` si los dos valores son iguales. perfecto para evitar divisiones por cero.

las 3 que vas a usar siempre 💔

CASE WHEN

segmentar, clasificar, crear buckets — cualquier pregunta que empiece con *"depende de..."*.

COALESCE

llenar huecos de NULL con un fallback. tu mejor amiga cuando los datos están sucios.

DATE_TRUNC

agrupar por mes, semana, día sin volver loca al GROUP BY. la base de cualquier reporte temporal.

11 window functions — *calcular sin perder filas*

calculan un valor para cada fila usando otras filas como contexto, *sin colapsar* el resultado como hace `GROUP BY` . lo más potente de SQL avanzado.

```sql
SELECT nombre, total,
  AVG(total) OVER () AS promedio_global,
  total - AVG(total) OVER ()
    AS diferencia_vs_promedio
FROM pedidos;
```

`OVER ()` vacío = calcula sobre todas las filas. cada pedido muestra su total *y* el promedio global.

```sql
SELECT nombre, pais, total,
  AVG(total) OVER (PARTITION BY pais)
    AS promedio_pais
FROM pedidos;
```

`PARTITION BY` = calcula por grupo, para cada fila. como `GROUP BY` pero sin colapsar.

```sql
SELECT nombre, fecha, total,
  SUM(total) OVER (ORDER BY fecha)
    AS acumulado
FROM pedidos;
```

suma acumulada: el total va creciendo fila por fila en orden de fecha.

FUNCIONES DE RANKING

`ROW_NUMBER()`	1, 2, 3, 4... sin empates
`RANK()`	1, 2, 2, 4... con saltos
`DENSE_RANK()`	1, 2, 2, 3... sin saltos
`NTILE(n)`	divide en n grupos
`LAG · LEAD`	fila anterior / siguiente

12 patrones clásicos — *queries que vas a usar siempre*

TOP N POR CATEGORÍA

```sql
WITH ranked AS (
  SELECT *,
    ROW_NUMBER() OVER (
      PARTITION BY categoria
      ORDER BY ventas DESC
    ) AS rn
  FROM productos
)
SELECT * FROM ranked WHERE rn <= 3;
```

DEDUPLICAR — MÁS RECIENTE POR EMAIL

```sql
WITH dedup AS (
  SELECT *, ROW_NUMBER() OVER (
    PARTITION BY email ORDER BY fecha DESC
  ) AS rn
  FROM clientes
)
SELECT * FROM dedup WHERE rn = 1;
```

PORCENTAJE SOBRE EL TOTAL

```sql
SELECT categoria,
  SUM(ventas) AS ventas_cat,
  ROUND(100.0 * SUM(ventas) /
    SUM(SUM(ventas)) OVER (), 1)
    AS porcentaje
FROM productos GROUP BY categoria;
```

PIVOT MANUAL · FILAS → COLUMNAS

```sql
SELECT
  SUM(CASE WHEN mes = 1 THEN ventas END) AS enero,
  SUM(CASE WHEN mes = 2 THEN ventas END) AS febrero,
  SUM(CASE WHEN mes = 3 THEN ventas END) AS marzo
FROM ventas;
```

13 errores comunes — *y cómo evitarlos*

ERROR	POR QUÉ PASA	SOLUCIÓN
`WHERE col = NULL`	`NULL` no es un valor — es ausencia. la comparación siempre falla.	usá `IS NULL` / `IS NOT NULL`
mezclar agr. con columnas	`SELECT nombre, COUNT(*)` sin `GROUP BY` falla porque `nombre` no está agregado.	agregá todas las columnas no-agregadas al `GROUP BY`
alias en WHERE	`WHERE` no puede usar alias del `SELECT` — se ejecuta antes.	repetí la expresión o usá un CTE / subquery
HAVING en vez de WHERE	usar `HAVING` para filtrar filas individuales es más lento.	filtrá con `WHERE` cuando no agrupás
JOIN sin ON	un JOIN sin condición crea un **producto cartesiano** (todas × todas).	siempre especificá `ON` con la clave correcta
dividir sin castear	`5/2 = 2` en entero. la división entera trunca el decimal.	usá `5.0/2` o `CAST(5 AS FLOAT)/2`
SELECT * en producción	trae columnas innecesarias, más lentas y frágiles ante cambios de esquema.	listá siempre las columnas que necesitás

14 referencia rápida — *estructura completa de una query*

una query SQL puede usar todas estas cláusulas. las marcadas con * son obligatorias.

```
WITH nombre_cte AS ( -- opcional: define tablas temporales SELECT ... ) * SELECT columnas, expresiones,
funciones * FROM tabla_principal JOIN otra_tabla ON condicion WHERE condicion_de_filas GROUP BY
columnas_de_agrupacion HAVING condicion_de_grupos ORDER BY columnas ASC / DESC LIMIT cantidad_de_filas
OFFSET filas_a_saltear;
```

CHECKLIST ANTES DE EJECUTAR

- ☑ todas las columnas del `SELECT` están en `GROUP BY` o son funciones de agregación
- ☑ los `JOIN` s tienen `ON` con la clave correcta
- ☑ usás `IS NULL` en vez de `= NULL`
- ☑ pusiste `LIMIT` para no traer millones de filas sin querer
- ☑ los alias no se usan en `WHERE` ni `HAVING` de la misma query
- ☑ las divisiones no pueden dar división por cero (usá `NULLIF` en el denominador)

PARA CERRAR

con **SELECT, WHERE, JOIN, GROUP BY** *y una window function encima, respondés el 95% de las preguntas de negocio que te van a hacer.*

data. pero cutie. — milo