

SELECT & FILTER

```
SELECT DISTINCT region, amount AS amt
FROM orders o
WHERE status = 'paid'
ORDER BY amt DESC, region
LIMIT 10 OFFSET 20;
```

= <> != < > <= >=	comparison
AND / OR / NOT	combine (use parentheses)
x IN ('a', 'b')	match any in list
x BETWEEN 1 AND 10	inclusive range
x LIKE 'ab%' / 'a_c'	% = any chars, _ = one char
x IS NULL / IS NOT NULL	never use = NULL
SELECT DISTINCT a, b	unique combinations
SELECT col AS alias	rename column or table

LOGICAL QUERY ORDER

FROM → JOIN → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY → LIMIT

WHERE filters rows before grouping; HAVING filters groups. Aliases from SELECT work in ORDER BY (and in SQLite/PostgreSQL GROUP BY) but not in WHERE.

AGGREGATES & GROUP BY

COUNT(*)	rows (including NULLs)
COUNT(col)	non-NULL values only
COUNT(DISTINCT col)	unique non-NULL values
SUM / AVG / MIN / MAX(col)	ignore NULLs

```
SELECT region, COUNT(*) AS n,
       ROUND(AVG(amount),1) AS avg_amt
FROM orders
WHERE status = 'paid'
GROUP BY region
HAVING COUNT(*) > 10
ORDER BY n DESC;
```

Every non-aggregated SELECT column must be in GROUP BY.

TIPS

Select only the columns you need; avoid `SELECT *` in production.

Check join row counts before and after: a many-to-many join silently multiplies rows.

Use CTEs to build queries step by step.

NULL = NULL is not true. Use IS NULL.

Without ORDER BY, row order is not guaranteed.

JOINS

INNER JOIN	only rows matching in both tables
LEFT JOIN	all left rows; NULLs where no match
RIGHT JOIN	all right rows (rare; swap tables instead)
FULL OUTER JOIN	all rows from both (not in older SQLite/MySQL)
CROSS JOIN	every combination (A x B)

```
SELECT c.name, o.amount
FROM customers c
LEFT JOIN orders o ON o.customer_id = c.id
WHERE o.id IS NULL; -- anti-join: no orders
```

Self join (employee and manager):

```
SELECT e.name AS emp, m.name AS boss
FROM staff e
LEFT JOIN staff m ON e.manager_id = m.id;
```

Filter the right table in `ON`, not `WHERE`, to keep LEFT JOIN rows.

COMBINE, SUBQUERY, CTE

q1 UNION q2	stack results, remove duplicates
q1 UNION ALL q2	stack, keep duplicates (faster)
INTERSECT / EXCEPT	common rows / rows only in q1

Same column count and compatible types required.

```
-- subquery
SELECT * FROM orders
WHERE amount > (SELECT AVG(amount) FROM orders);
WHERE id IN (SELECT customer_id FROM vip);

-- CTE (named, readable subquery)
WITH monthly AS (
  SELECT strftime('%Y-%m', order_date) AS ym,
         SUM(amount) AS rev
  FROM orders GROUP BY ym
)
SELECT * FROM monthly WHERE rev > 1000;
```

CASE, NULL HANDLING, CAST

```
CASE WHEN amount >= 100 THEN 'big'
      WHEN amount >= 50  THEN 'mid'
      ELSE 'small' END AS size
```

COALESCE(a, b, 0)	first non-NULL value
NULLIF(a, 0)	NULL if a = 0 (avoids divide by zero)
x / NULLIF(y, 0)	safe division
CAST(x AS INTEGER)	convert type (also REAL, TEXT, DATE)
SUM(CASE WHEN c THEN 1 ELSE 0 END)	conditional count

Integer / integer truncates in most databases; use `1.0 * a / b`.

DATES (DIFFER BY DATABASE!)

SQLite

strftime('%Y-%m', d)	year-month text
strftime('%w', d)	weekday 0=Sun; %j day of year
date(d, 'start of month')	first day of month
date(d, '+7 days')	date arithmetic
julianday(b) - julianday(a)	days between
date('now')	today

PostgreSQL

```
DATE_TRUNC('month', order_date)
EXTRACT(YEAR FROM order_date)
order_date + INTERVAL '7 days'
CURRENT_DATE - order_date -- days
```

MySQL: `DATE_FORMAT`, `DATEDIFF`. SQL Server: `DATEPART`, `DATEDIFF`.

STRING FUNCTIONS

a b	concatenate (MySQL: CONCAT)
LENGTH(s)	number of characters
UPPER(s) / LOWER(s)	change case
TRIM(s)	strip spaces both ends
SUBSTR(s, start, len)	substring, 1-based
INSTR(s, 'x')	position of text (0 = none)
REPLACE(s, 'a', 'b')	replace all matches

WINDOW FUNCTIONS

`func() OVER (PARTITION BY g ORDER BY x)` keeps every row (no collapse).

ROW_NUMBER()	1,2,3,4 unique sequence
RANK()	1,2,2,4 ties share, gaps
DENSE_RANK()	1,2,2,3 ties share, no gaps
LAG(x, 1) / LEAD(x, 1)	previous / next row value
NTILE(4)	split into 4 buckets

```
-- running total
SUM(amount) OVER (
  PARTITION BY customer_id
  ORDER BY order_date) AS running
-- 7-row moving avg
AVG(amount) OVER (
  ORDER BY order_date
  ROWS BETWEEN 6 PRECEDING AND CURRENT ROW)
```

Frames: `ROWS BETWEEN n PRECEDING AND CURRENT ROW`, or `UNBOUNDED PRECEDING`. Not allowed in WHERE: use a CTE.

COMMON ANALYST QUERIES

Top N per group

```
WITH r AS (
  SELECT *, ROW_NUMBER() OVER (
    PARTITION BY region ORDER BY amount DESC) rn
  FROM orders)
SELECT * FROM r WHERE rn <= 3;
```

Month-over-month % change

```
WITH m AS (
  SELECT strftime('%Y-%m', order_date) AS ym,
         SUM(amount) AS rev
  FROM orders GROUP BY ym)
SELECT ym, rev,
       100.0 * (rev - LAG(rev) OVER (ORDER BY ym))
       / LAG(rev) OVER (ORDER BY ym) AS mom_pct
FROM m;
```

Find duplicates

```
SELECT email, COUNT(*) FROM customers
GROUP BY email HAVING COUNT(*) > 1;
```

Retention: self join activity on user_id and month = prior month; count matched users / month-1 users.

SQL FROM PYTHON

```
import sqlite3, pandas as pd
con = sqlite3.connect("data.db")
df = pd.read_sql_query(
  "SELECT * FROM orders WHERE amount > ?",
  con, params=(100,))
df.to_sql("clean", con, if_exists="replace",
         index=False)
```

Use `?` placeholders, not f-strings (SQL injection).