VIRTAVIS LEARNING

FREE CHEAT SHEET

SQL for Data Analytics

The queries you need to filter, summarise, join and rank data.
PostgreSQL syntax, tested in PostgreSQL 16.

THE ORDER SQL RUNS IN

`FROM` › `WHERE` › `GROUP BY` › `HAVING` › `SELECT` › `ORDER BY` › `LIMIT`

First We Group, Have Selections Ordered Later

GOLDEN RULE

SELECT runs late, so WHERE can't use a column alias. Repeat the expression or use a CTE.

ANATOMY OF A QUERY

Write the clauses in this order.

```
SELECT city, COUNT(*) AS customers  -- columns
FROM customers                      -- table
WHERE signup_date >= '2024-01-01'   -- filter rows
GROUP BY city                       -- groups
HAVING COUNT(*) > 10                -- filter groups
ORDER BY customers DESC             -- sort
LIMIT 10;                           -- how many
```

Only SELECT and FROM are required. SQL Server uses SELECT TOP 10 instead of LIMIT.

SELECT COLUMNS

Choose what comes back.

Some columns
```
SELECT customer_id, customer_name
FROM customers;
```

All columns
```
SELECT * FROM customers;
```

Rename
```
SELECT amount AS order_value
FROM orders;
```

Calculate
```
SELECT amount * 1.2 AS amount_inc_vat
FROM orders;
```

Unique values
```
SELECT DISTINCT city FROM customers;
```

SORT AND LIMIT

Order and trim the result.

Two columns
```
SELECT * FROM orders
ORDER BY region ASC, amount DESC;
```
ASC (low to high) is the default

Top 10
```
SELECT * FROM orders
ORDER BY amount DESC
LIMIT 10;
```
Always ORDER BY before LIMIT

FIVE CLASSIC MISTAKES

Check these before you share a result.

1. **WHERE vs HAVING.** WHERE filters rows before grouping, HAVING filters groups after.
2. **NULL is not a value.** Use IS NULL. COUNT(email) skips NULLs; COUNT(*) doesn't.
3. **Whole-number division.** 5 / 2 = 2. Write 100.0 * part / total.
4. **Joins can add rows.** Check COUNT(*) before and after every join.
5. **AND before OR.** a AND b OR c means (a AND b) OR c. Use brackets.

FILTER ROWS

WHERE keeps rows that match.

Equal / not equal
```
WHERE city = 'London'
WHERE status <> 'Cancelled'
```
<> and != both work. Text goes in single quotes

Compare
```
WHERE amount > 100
WHERE amount <= 100
```

AND / OR
```
WHERE region = 'UK'
  AND (amount > 100 OR status = 'VIP')
```
AND runs before OR, so add brackets

In a list
```
WHERE region IN ('UK', 'US')
```
NOT IN ('UK', 'US') excludes them

Range
```
WHERE amount BETWEEN 100 AND 500
```
Both ends included

Pattern
```
WHERE customer_name LIKE 'A%'
```
% any text, _ one character. ILIKE ignores case

Missing
```
WHERE email IS NULL
```
IS NOT NULL for the opposite. = NULL never matches

SUMMARISE AND GROUP

Aggregate functions turn many rows into one.

Count rows
```
COUNT(*)
```
COUNT(email) skips NULLs

Count unique
```
COUNT(DISTINCT customer_id)
```

Total, average
```
SUM(amount)   AVG(amount)
```

Lowest, highest
```
MIN(order_date)   MAX(order_date)
```

One row per region

```
SELECT region,
  COUNT(*) AS orders,
  SUM(amount) AS total_sales,
  ROUND(AVG(amount)) AS avg_order
FROM orders
GROUP BY region
ORDER BY total_sales DESC;
```

region	orders	total_sales	avg_order
UK	1,250	250,000	200
US	980	180,000	184
CA	760	120,000	158

Every column in SELECT is either in GROUP BY or inside an aggregate.

FILTER GROUPS

HAVING works on totals, after GROUP BY.

```
SELECT region, SUM(amount) AS total_sales
FROM orders
GROUP BY region
HAVING SUM(amount) > 150000;
```

region	total_sales
UK	250,000
US	180,000

CA (120,000) is dropped. WHERE can't do this: totals don't exist yet.

JOIN TABLES

Combine tables on a shared key.



Every order, with the customer's name

```
SELECT o.order_id, o.customer_id,
  c.customer_name, o.amount
FROM orders o
LEFT JOIN customers c
  ON o.customer_id = c.customer_id;
```

orders

order_id	customer_id	amount
1	1	100
2	2	200
3	3	150

+ customers

customer_id	customer_name
1	Amy
2	Ben
4	Dan

=

order_id	customer_id	customer_name	amount
1	1	Amy	100
2	2	Ben	200
3	3	NULL	150

Customer 3 isn't in customers, so the name is NULL. INNER JOIN would drop order 3.

Customers with no orders join from customers, keep the gaps

```
SELECT c.customer_id, c.customer_name
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;
```

IF / THEN LOGIC

CASE WHEN creates a new column from rules.

```
SELECT amount,
  CASE WHEN amount >= 200 THEN 'High'
       WHEN amount >= 150 THEN 'Medium'
       ELSE 'Low'
  END AS segment
FROM orders;
```

amount	segment
100	Low
200	High
150	Medium

Checked top to bottom and the first match wins, so put the biggest threshold first.

WORK WITH DATES

PostgreSQL date functions.

Year / month
```
EXTRACT(YEAR FROM order_date)
EXTRACT(MONTH FROM order_date)
```

Start of month
```
DATE_TRUNC('month', order_date)
```
Groups dates by month

Date range
```
WHERE order_date >= '2024-01-01'
  AND order_date < '2025-01-01'
```
Safer than BETWEEN when times are stored

Days between
```
order_date - signup_date
```
Two dates give a whole number of days

Add a month
```
order_date + INTERVAL '1 month'
```

SUBQUERIES AND CTES

Break a hard question into steps.

Subquery orders above the average

```
SELECT order_id, amount
FROM orders
WHERE amount > (SELECT AVG(amount) FROM orders);
```

CTE name a step with WITH, then query it. Easier to check

```
WITH monthly AS (
  SELECT DATE_TRUNC('month', order_date) AS month,
         SUM(amount) AS total
  FROM orders GROUP BY 1
)
SELECT * FROM monthly ORDER BY month;
```

WINDOW FUNCTIONS

Calculate across rows, keep every row.

Number rows
```
ROW_NUMBER() OVER (ORDER BY amount DESC)
```

Rank in group
```
RANK() OVER (PARTITION BY region
  ORDER BY amount DESC)
```
PARTITION BY restarts per group

Previous row
```
LAG(amount) OVER (ORDER BY order_date)
```
LEAD() looks at the next row

Running total
```
SUM(amount) OVER (ORDER BY order_date
  ROWS UNBOUNDED PRECEDING)
```

Ties three ways to rank

```
SELECT name, amount,
  ROW_NUMBER() OVER w AS row_num,
  RANK()       OVER w AS rnk,
  DENSE_RANK() OVER w AS dense
FROM scores
WINDOW w AS (ORDER BY amount DESC);
```

name	amount	row_num	rnk	dense
Amy	900	1	1	1
Ben	900	2	1	1
Cara	700	3	3	2

To filter on a rank, work it out in a CTE first.

COMMON BUSINESS QUESTIONS

Ready-to-adapt queries.

Total revenue
```
SELECT SUM(amount) FROM orders
WHERE status <> 'Cancelled';
```

Revenue by month 1 = the first column in SELECT
```
SELECT DATE_TRUNC('month', order_date) AS month,
  SUM(amount) AS revenue
FROM orders GROUP BY 1 ORDER BY 1;
```

% cancelled 100.0 avoids whole-number division
```
SELECT ROUND(100.0 * COUNT(*) FILTER
  (WHERE status = 'Cancelled') / COUNT(*), 1)
FROM orders;
```

Check the numbers. Then say what they mean.

By **Sophie Lumpa**, Data and Business Intelligence Specialist. Every query tested in PostgreSQL 16. © 2026 Virtavis Learning · Free to share with credit

Start the course › SQL for Data Analytics

VIRTAVIS
virtavis.com/courses