Absolutely, brochacho 💪 — here’s your **PostgreSQL SQL Cheat Sheet** 🔥  
All the **basic commands** you’ll need as a developer — neat, clean, and copy-paste ready!

## 🧾 POSTGRESQL SQL CHEAT SHEET

### 🔍 SELECT Statements

sql

CopyEdit

SELECT \* FROM table\_name; -- all columns

SELECT column1, column2 FROM table\_name; -- specific columns

### 🚫 DISTINCT

sql

CopyEdit

SELECT DISTINCT column\_name FROM table\_name;

### 📊 ORDER BY

sql

CopyEdit

SELECT \* FROM table\_name ORDER BY column\_name ASC; -- Ascending

SELECT \* FROM table\_name ORDER BY column\_name DESC; -- Descending

### 🎯 WHERE Clause + Comparison Operators

sql

CopyEdit

-- Equality and Inequality

SELECT \* FROM table\_name WHERE column = 'value';

SELECT \* FROM table\_name WHERE column != 'value';

SELECT \* FROM table\_name WHERE column <> 'value'; -- same as !=

-- Greater / Lesser Than

SELECT \* FROM table\_name WHERE column > 100;

SELECT \* FROM table\_name WHERE column >= 100;

SELECT \* FROM table\_name WHERE column < 100;

SELECT \* FROM table\_name WHERE column <= 100;

-- Range

SELECT \* FROM table\_name WHERE column BETWEEN 10 AND 50;

-- Null check

SELECT \* FROM table\_name WHERE column IS NULL;

SELECT \* FROM table\_name WHERE column IS NOT NULL;

### 🧠 Logical Operators

sql

CopyEdit

-- AND, OR, NOT

SELECT \* FROM table\_name WHERE column1 = 'A' AND column2 = 'B';

SELECT \* FROM table\_name WHERE column1 = 'A' OR column2 = 'B';

SELECT \* FROM table\_name WHERE NOT column1 = 'A';

### 🎛️ GROUP BY & Aggregates

sql

CopyEdit

SELECT column, COUNT(\*) FROM table\_name GROUP BY column;

SELECT department, AVG(salary)

FROM employees

GROUP BY department;

### 🎯 HAVING (filtering aggregated results)

sql

CopyEdit

SELECT department, COUNT(\*)

FROM employees

GROUP BY department

HAVING COUNT(\*) > 5;

### 🧮 Aggregate Functions

sql

CopyEdit

SELECT COUNT(\*) FROM table\_name;

SELECT SUM(salary) FROM employees;

SELECT AVG(salary) FROM employees;

SELECT MIN(salary) FROM employees;

SELECT MAX(salary) FROM employees;

### 🧱 INSERT

sql

CopyEdit

INSERT INTO table\_name (column1, column2)

VALUES ('value1', 'value2');

### ✏️ UPDATE

sql

CopyEdit

UPDATE table\_name

SET column1 = 'new\_value'

WHERE column2 = 'some\_condition';

### ❌ DELETE

sql

CopyEdit

DELETE FROM table\_name

WHERE column = 'value';

### 🔗 JOINs (based on common columns)

#### INNER JOIN

sql

CopyEdit

SELECT a.\*, b.\*

FROM table\_a a

INNER JOIN table\_b b

ON a.id = b.a\_id;

#### LEFT JOIN

sql

CopyEdit

SELECT a.\*, b.\*

FROM table\_a a

LEFT JOIN table\_b b

ON a.id = b.a\_id;

#### RIGHT JOIN

sql

CopyEdit

SELECT a.\*, b.\*

FROM table\_a a

RIGHT JOIN table\_b b

ON a.id = b.a\_id;

#### FULL JOIN

sql

CopyEdit

SELECT a.\*, b.\*

FROM table\_a a

FULL OUTER JOIN table\_b b

ON a.id = b.a\_id;

### 🧪 LIKE / ILIKE (pattern matching)

sql

CopyEdit

SELECT \* FROM table\_name WHERE column LIKE 'A%'; -- starts with A

SELECT \* FROM table\_name WHERE column LIKE '%A'; -- ends with A

SELECT \* FROM table\_name WHERE column LIKE '%A%'; -- contains A

SELECT \* FROM table\_name WHERE column ILIKE '%a%'; -- case-insensitive

### 🎲 LIMIT & OFFSET

sql

CopyEdit

SELECT \* FROM table\_name LIMIT 10;

SELECT \* FROM table\_name LIMIT 10 OFFSET 20;

### 🔎 Subqueries

sql

CopyEdit

SELECT name

FROM employees

WHERE department\_id IN (SELECT id FROM departments WHERE name = 'Engineering');

### 💣 DROP, TRUNCATE, DELETE

sql

CopyEdit

DROP TABLE table\_name; -- delete structure + data

TRUNCATE TABLE table\_name; -- delete all data, keep structure

DELETE FROM table\_name; -- delete rows (can use WHERE)

### 🛠️ CREATE TABLE (Basic)

sql

CopyEdit

CREATE TABLE users (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

age INT,

created\_at TIMESTAMP DEFAULT NOW()

);

### 🧵 BONUS: Combine filters from multiple tables

sql

CopyEdit

-- Employees with salary >= 50k and in 'Engineering' department

SELECT e.name, e.salary, d.name as department

FROM employees e

JOIN departments d ON e.department\_id = d.id

WHERE e.salary >= 50000 AND d.name = 'Engineering';

can I write a query on a table with users and there tickets with these basic commands where I have to select all the users with more han 3 tickets in the past month can I do that??

###### ChatGPT said:

Yes bro 🔥 — you can **100% do that** using basic PostgreSQL commands like:

* JOIN
* WHERE
* GROUP BY
* HAVING
* date functions like CURRENT\_DATE or NOW()

Let’s break it down step by step 👇

## ✅ Problem

You have:

* A users table: contains user info (id, name, etc.)
* A tickets table: contains ticket info (id, user\_id, created\_at, etc.)

And you want to:

“Select all users who created **more than 3 tickets in the past month**”

## 🔍 SQL Query

sql

CopyEdit

SELECT u.id, u.name, COUNT(t.id) AS ticket\_count

FROM users u

JOIN tickets t ON u.id = t.user\_id

WHERE t.created\_at >= CURRENT\_DATE - INTERVAL '1 month'

GROUP BY u.id, u.name

HAVING COUNT(t.id) > 3;

### 💥 Explanation

| Part | What it does |
| --- | --- |
| JOIN tickets ON u.id = t.user\_id | Connects tickets to users |
| WHERE t.created\_at >= CURRENT\_DATE - INTERVAL '1 month' | Filters only tickets from the last 30 days |
| GROUP BY u.id, u.name | Groups results by user |
| COUNT(t.id) | Counts tickets per user |
| HAVING COUNT(t.id) > 3 | Keeps only users with more than 3 tickets |

### ✅ Sample Table Data

**users**

| id | name |
| --- | --- |
| 1 | Vishnu |
| 2 | Yash |

**tickets**

| id | user\_id | created\_at |
| --- | --- | --- |
| 1 | 1 | 2025-06-01 |
| 2 | 1 | 2025-06-10 |
| 3 | 1 | 2025-06-15 |
| 4 | 1 | 2025-06-20 |
| 5 | 2 | 2025-06-05 |

This will return only Vishnu because he has **4 tickets in the past month**.

Let me know if your created\_at field uses timestamps instead of dates — I can adjust it to NOW() or DATE\_TRUNC.

##### You said:

like not in a single query but writing 2 queries like we get all the uids first and then search the user table with those??

###### ChatGPT said:

Ahhh okay, I see what you’re asking now, bro — you want to **break it into 2 simpler queries**, not do it all in one shot.

✅ Yes! You **can totally** do that using a **subquery or two-step process**.

## 🧠 Goal (in 2 steps):

Step 1: Get all user\_ids who have more than 3 tickets in the last month  
Step 2: Use those IDs to fetch user details from the users table

## 🔹 ****Step 1**** — Get qualifying user\_ids

sql

CopyEdit

SELECT user\_id

FROM tickets

WHERE created\_at >= CURRENT\_DATE - INTERVAL '1 month'

GROUP BY user\_id

HAVING COUNT(\*) > 3;

Let’s say this returns:

diff

CopyEdit

user\_id

--------

1

4

7

## 🔹 ****Step 2**** — Get user details from users using those user\_ids

sql

CopyEdit

SELECT \*

FROM users

WHERE id IN (

SELECT user\_id

FROM tickets

WHERE created\_at >= CURRENT\_DATE - INTERVAL '1 month'

GROUP BY user\_id

HAVING COUNT(\*) > 3

);

OR (if you're doing it manually, outside of SQL):

1. First run **Step 1** and get the IDs like: 1, 4, 7
2. Then run:

sql

CopyEdit

SELECT \* FROM users WHERE id IN (1, 4, 7);

### ✅ This is called a ****subquery**** approach (or two-step if done manually), and it's great when:

* You want to debug step by step
* You're testing intermediate results
* You want readable logic instead of a monster query

Let me know if you want a version using **temporary tables** or **CTEs (**WITH **clause)** too.

Tools

ChatGPT can make mistakes. Check important info. Se

**select** make,model,**min**(price) **from** car **group** **by** make,model **order** **by** **min**(price) **desc**;

the min(price) be used as a column so all the column operations like order by can be used