# **PostgreSQL SQL Guide**

## **1. Introduction to PostgreSQL**

PostgreSQL (often called Postgres) is an **open-source relational database management system (RDBMS)** known for its robust features, extensibility, and compliance with SQL standards.

**Key Features of PostgreSQL:**

* ACID compliance (Atomicity, Consistency, Isolation, Durability)
* Support for JSON, XML, and unstructured data
* Extensibility (custom functions, operators, and procedural languages)
* Full-text search, GIS support with PostGIS

## **2. Database and Table Management**

### **2.1 Creating a Database**

To create a new database in PostgreSQL:

sql

CREATE DATABASE company\_db;

To connect to the database:

bash

psql -d company\_db;

To list all databases:

sql

\l

To drop a database:

sql

DROP DATABASE company\_db;

### **2.2 Creating a Table**

Tables store structured data in **rows and columns**.

sql

CREATE TABLE employees (

id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

age INT CHECK (age > 18),

salary DECIMAL(10,2),

department VARCHAR(50),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Explanation:**

* SERIAL PRIMARY KEY: Auto-incrementing ID.
* VARCHAR(50): String of up to 50 characters.
* UNIQUE: Ensures unique values in the column.
* CHECK (age > 18): Ensures age is greater than 18.
* DEFAULT CURRENT\_TIMESTAMP: Automatically adds the timestamp when a row is inserted.

### **2.3 Modifying a Table**

**Add a new column:** sql  
  
ALTER TABLE employees ADD COLUMN phone\_number VARCHAR(20);

**Rename a column:** sql  
  
ALTER TABLE employees RENAME COLUMN phone\_number TO contact\_number;

**Remove a column:** sql  
  
ALTER TABLE employees DROP COLUMN contact\_number;

**Change data type:** sql  
  
ALTER TABLE employees ALTER COLUMN salary TYPE FLOAT;

### **2.4 Deleting a Table**

sql

DROP TABLE employees;

## **3. Basic SQL Queries**

### **3.1 Inserting Data**

sql

INSERT INTO employees (first\_name, last\_name, email, age, salary, department)

VALUES ('John', 'Doe', 'john.doe@example.com', 30, 50000.00, 'IT');

**Inserting multiple rows:**

sql

INSERT INTO employees (first\_name, last\_name, email, age, salary, department)

VALUES

('Alice', 'Smith', 'alice.smith@example.com', 28, 55000, 'HR'),

('Bob', 'Brown', 'bob.brown@example.com', 35, 60000, 'Finance');

### **3.2 Retrieving Data**

**Select all columns:** sql  
  
SELECT \* FROM employees;

**Select specific columns:** sql  
  
SELECT first\_name, last\_name, department FROM employees;

**Filter using WHERE:** sql  
  
SELECT \* FROM employees WHERE department = 'IT';

**Find employees earning more than 50,000:** sql  
  
SELECT \* FROM employees WHERE salary > 50000;

### **3.3 Updating Data**

sql

UPDATE employees

SET salary = 70000

WHERE email = 'john.doe@example.com';

Updating multiple columns:

sql

UPDATE employees

SET salary = 75000, department = 'Management'

WHERE id = 1;

### **3.4 Deleting Data**

sql

DELETE FROM employees WHERE email = 'john.doe@example.com';

Deleting all rows:

sql

DELETE FROM employees;

**(Caution: This removes all data but keeps the table structure.)**

## **4. Advanced SQL Queries**

### **4.1 Sorting Data**

sql

SELECT \* FROM employees ORDER BY salary DESC;

Sorting by multiple columns:

sql

SELECT \* FROM employees ORDER BY department ASC, salary DESC;

### **4.2 Aggregation Functions**

sql

SELECT COUNT(\*) FROM employees;

SELECT AVG(salary) FROM employees;

SELECT MAX(salary) FROM employees;

SELECT MIN(age) FROM employees;

Grouping data:

sql

SELECT department, AVG(salary) FROM employees GROUP BY department;

### **4.3 Using Joins**

#### **Inner Join**

sql

SELECT employees.first\_name, employees.last\_name, departments.department\_name

FROM employees

INNER JOIN departments ON employees.department = departments.department\_id;

#### **Left Join**

sql

SELECT employees.first\_name, employees.last\_name, departments.department\_name

FROM employees

LEFT JOIN departments ON employees.department = departments.department\_id;

#### **Right Join**

sql

SELECT employees.first\_name, employees.last\_name, departments.department\_name

FROM employees

RIGHT JOIN departments ON employees.department = departments.department\_id;

### **4.4 Subqueries**

Find employees earning above the average salary:

sql

SELECT \* FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);

### **4.5 Common Table Expressions (CTE)**

sql

WITH HighSalary AS (

SELECT \* FROM employees WHERE salary > 60000

)

SELECT \* FROM HighSalary;

### **4.6 Window Functions**

Running total of salaries:

sql

SELECT first\_name, salary, SUM(salary) OVER (ORDER BY id) AS running\_total FROM employees;

Rank employees by salary:

sql

SELECT first\_name, salary, RANK() OVER (ORDER BY salary DESC) AS rank FROM employees;

### **4.7 Transactions**

sql

BEGIN;

UPDATE employees SET salary = salary \* 1.1 WHERE department = 'IT';

COMMIT;

To rollback:

sql

ROLLBACK;

## **5. Indexing for Performance**

Creating an index on email for faster lookups:

sql

CREATE INDEX idx\_email ON employees(email);

## **6. JSON Support**

Storing JSON data:

sql

CREATE TABLE products (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

details JSONB

);

Inserting JSON data:

sql

INSERT INTO products (name, details) VALUES

('Laptop', '{"brand": "Dell", "RAM": "16GB", "SSD": "512GB"}');

Querying JSON:

sql

SELECT name, details->>'brand' AS brand FROM products;

## **7. Full-Text Search**

sql

SELECT \* FROM employees WHERE to\_tsvector(first\_name || ' ' || last\_name) @@ to\_tsquery('John');

## **8. User Management**

Creating a user:

sql

CREATE USER dev\_user WITH PASSWORD 'securepassword';

GRANT ALL PRIVILEGES ON DATABASE company\_db TO dev\_user;

## **9. Backup and Restore**

Backup:

bash

pg\_dump company\_db > backup.sql

Restore:

bash

psql company\_db < backup.sql