

PostgreSQL Quick Reference

Tables, queries, joins, indexes, JSON, roles

Connecting

Command Line

```
psql -U postgres
psql -h localhost -p 5432 -U user -d mydb
psql "postgres://user:pass@host:5432/mydb"
```

psql Meta-Commands

```
\l          List databases
\c dbname   Connect to database
\dt         List tables
\d tablename Describe table structure
\dn         List schemas
\du         List roles
\q         Quit psql
\i file.sql Execute SQL file
```

Tables & Schemas

Create Table

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email TEXT UNIQUE,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

Schema Operations

```
CREATE SCHEMA app;
CREATE TABLE app.users (id SERIAL PRIMARY KEY);
SET search_path TO app, public;
DROP SCHEMA app CASCADE;
```

Alter Table

```
ALTER TABLE users ADD COLUMN age INT;
ALTER TABLE users ALTER COLUMN name TYPE TEXT;
ALTER TABLE users DROP COLUMN age;
ALTER TABLE users RENAME TO customers;
```

Data Types

Numeric

INTEGER / INT	4-byte integer
BIGINT	8-byte integer
SERIAL	Auto-incrementing integer
NUMERIC(p, s)	Exact numeric (e.g., NUMERIC(10,2))
REAL / DOUBLE PRECISION	Floating-point (4 / 8 bytes)
BOOLEAN	true / false / null

String & Binary

TEXT	Variable unlimited text
VARCHAR(n)	Variable text up to n chars
CHAR(n)	Fixed-length text
BYTEA	Binary data
UUID	128-bit universally unique id

Date, JSON & Array

DATE	Calendar date
TIMESTAMPTZ	Timestamp with time zone
INTERVAL	Time span (e.g., '2 days')
JSONB	Binary JSON (indexable)
INT[] / TEXT[]	Array types

Queries

Insert

```
INSERT INTO users (name, email)
VALUES ('Alice', 'alice@example.com')
RETURNING id;

INSERT INTO users (name, email) VALUES
('Bob', 'bob@ex.com'),
('Carol', 'carol@ex.com');
```

Select

```
SELECT * FROM users WHERE id = 1;
SELECT name, email FROM users
ORDER BY name LIMIT 10 OFFSET 20;
```

Update

```
UPDATE users SET email = 'new@ex.com'
WHERE id = 1 RETURNING *;
```

Upsert

```
INSERT INTO users (email, name)
VALUES ('a@ex.com', 'Alice')
ON CONFLICT (email) DO UPDATE
SET name = EXCLUDED.name;
```

Delete

```
DELETE FROM users WHERE id = 1 RETURNING *;
TRUNCATE TABLE users RESTART IDENTITY;
```

Joins & Subqueries

Join Types

INNER JOIN	Rows matching in both tables
LEFT JOIN	All left rows + matching right
RIGHT JOIN	All right rows + matching left
FULL OUTER JOIN	All rows from both tables
CROSS JOIN	Cartesian product
LATERAL JOIN	Subquery referencing outer row

CTE (Common Table Expression)

```
WITH active AS (
  SELECT * FROM users WHERE active = true
)
SELECT a.name, o.total
FROM active a
JOIN orders o ON a.id = o.user_id;
```

Subquery

```
SELECT name FROM users
WHERE id IN (
  SELECT user_id FROM orders
  WHERE total > 100
);
```

Indexes

Create & Drop

```
CREATE INDEX idx_name ON users(name);
CREATE UNIQUE INDEX idx_email ON users(email);
CREATE INDEX idx_gin ON posts USING GIN(tags);
DROP INDEX idx_name;
```

Index Types

B-tree	Default, good for =, <, >, BETWEEN
Hash	Equality comparisons only
GIN	Generalized inverted — arrays, JSONB, full-text
GiST	Generalized search — geometric, range
BRIN	Block range — large sorted tables

Query Analysis

```
EXPLAIN ANALYZE
SELECT * FROM users WHERE name = 'Alice';
```

Functions & Procedures

SQL Function

```
CREATE FUNCTION active_count()
RETURNS INTEGER AS $$
SELECT COUNT(*)::INT FROM users
WHERE active = true;
$$ LANGUAGE sql;
SELECT active_count();
```

PL/pgSQL Function

```
CREATE FUNCTION greet(name TEXT)
RETURNS TEXT AS $$
BEGIN
  RETURN 'Hello, ' || name;
END;
$$ LANGUAGE plpgsql;
```

Useful Built-ins

NOW() / CURRENT_TIMESTAMP	Current timestamp with TZ
AGE(ts1, ts2)	Interval between timestamps
COALESCE(a, b)	First non-null value
NULLIF(a, b)	NULL if a=b
GENERATE_SERIES(1, 10)	Generate rows of sequential values
STRING_AGG(col, ',')	Concatenate values with separator

Roles & Permissions

Role Management

```
CREATE ROLE app LOGIN PASSWORD 'secret';
ALTER ROLE app SET search_path TO myapp;
DROP ROLE app;
```

Grants

```
GRANT ALL ON DATABASE mydb TO app;
GRANT SELECT, INSERT ON users TO reader;
GRANT USAGE ON SCHEMA public TO app;
REVOKE INSERT ON users FROM reader;
```

Row-Level Security

```
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
CREATE POLICY user_own ON users
FOR ALL USING (id = current_setting('app.uid')::INT);
```

JSON Support

JSONB Operators

-> 'key'	Get JSON value by key (as JSON)
->> 'key'	Get JSON value by key (as text)
#> '{a, b}'	Get nested value by path
@>	Contains (left includes right)
?	Key exists
 	Concatenate JSONB values

PostgreSQL Quick Reference

JSONB Queries

```
SELECT data->>'name' FROM profiles
WHERE data @> '{"active": true}';
```

```
SELECT * FROM profiles
WHERE data ? 'email';
```

JSONB Functions

```
SELECT jsonb_each(data) FROM profiles;
SELECT jsonb_array_elements('[1,2,3]');
SELECT jsonb_set(data, '{name}', 'Alice')
FROM profiles WHERE id = 1;
```

Common Patterns

Transactions

```
BEGIN;
UPDATE accounts SET balance = balance - 100
WHERE id = 1;
UPDATE accounts SET balance = balance + 100
WHERE id = 2;
COMMIT; -- or ROLLBACK;
```

Window Functions

```
SELECT name, salary,
       RANK() OVER (ORDER BY salary DESC),
       AVG(salary) OVER (PARTITION BY dept)
FROM employees;
```

Copy Data

```
COPY users TO '/tmp/users.csv'
WITH (FORMAT csv, HEADER);
COPY users FROM '/tmp/users.csv'
WITH (FORMAT csv, HEADER);
```

pg_dump Backup

```
pg_dump -U postgres mydb > backup.sql
pg_dump -Fc mydb > backup.dump
pg_restore -d mydb backup.dump
```