PostgreSQL Cheatsheet

Essential commands and operations for database management and SQL queries

This cheatsheet provides a quick reference to fundamental PostgreSQL operations, syntax, and advanced features, ideal for both beginners and experienced developers for efficient database management and SQL development.

Connection & Setup — Connect to databases and configure psql

Database Management — Create, modify, and manage databases

Table Operations — Create and manipulate table structures

Data Querying — Query and retrieve data from tables

Performance & Admin — Optimize queries and manage server

Connection & Database Setup

Connect to PostgreSQL: `psql`

Connect to a local or remote PostgreSQL database using psql command-line tool.

```
# Connect to local database
psql -U username -d database_name

# Connect to remote database
psql -h hostname -p 5432 -U username -d
database_name

# Connect with password prompt
psql -U postgres -W

# Connect using connection string
psql "host=localhost port=5432 dbname=mydb
user=myuser"
```

Create Database: `CREATE DATABASE`

Create a new database in PostgreSQL using the CREATE DATABASE command.

```
# Create a new database
CREATE DATABASE mydatabase;

# Create database with owner
CREATE DATABASE mydatabase OWNER myuser;

# Create database with encoding
CREATE DATABASE mydatabase
  WITH ENCODING 'UTF8'
  LC_COLLATE='en_US.UTF-8'
  LC_CTYPE='en_US.UTF-8';
```

List Databases: `\l`

List all databases in the PostgreSQL server.

```
# List all databases
\l

# List databases with detailed info
\l+

# Connect to different database
\c database_name
```

Basic psql Commands

Essential psql terminal commands for navigation and information.

```
# Quit psql
\q

# Get help for SQL commands
\help CREATE TABLE

# Get help for psql commands
\?

# Show current database and user
\conninfo

# Execute system commands
\! ls
```

Database Information: `\d`

Display database objects and their detailed information.

```
# List all tables
\dt

# List all tables with details
\dt+

# Describe specific table
\d table_name

# List all schemas
\dn

# List all users/roles
\du
```

Version & Settings

Check PostgreSQL version and configuration settings.

```
# Check PostgreSQL version
SELECT version();

# Show current settings
SHOW ALL;

# Show specific setting
SHOW max_connections;

# Set configuration parameter
SET work_mem = '256MB';
```

Table Creation & Management

Create and modify table structures with appropriate data types and constraints.

01

Create Table: `CREATE TABLE`

Define new tables with columns, data types, and constraints.

```
# Basic table creation
CREATE TABLE users (
   id SERIAL PRIMARY KEY,
   username VARCHAR(50) NOT
NULL UNIQUE,
   email VARCHAR(100) NOT NULL,
   created_at TIMESTAMP DEFAULT
NOW()
);

# Table with foreign key
CREATE TABLE orders (
   id SERIAL PRIMARY KEY,
   user_id INTEGER REFERENCES
users(id),
   total DECIMAL(10,2) NOT NULL,
   status VARCHAR(20) DEFAULT
'pending'
);
```

02

Modify Tables: `ALTER TABLE`

Add, modify, or remove columns and constraints from existing tables.

```
# Add new column
ALTER TABLE users ADD COLUMN
phone VARCHAR(15);

# Change column type
ALTER TABLE users ALTER COLUMN
phone TYPE VARCHAR(20);

# Drop column
ALTER TABLE users DROP COLUMN
phone;

# Add constraint
ALTER TABLE users ADD
CONSTRAINT unique_email
   UNIQUE (email);
```

03

Drop & Truncate: `DROP/TRUNCATE`

Remove tables or clear all data from tables.

```
# Drop table completely
DROP TABLE IF EXISTS old_table;

# Remove all data but keep structure
TRUNCATE TABLE users;

# Truncate with restart identity
TRUNCATE TABLE users RESTART
IDENTITY;
```

Data Types & Constraints

Common Data Types

Essential PostgreSQL data types for different kinds of data.

```
# Numeric types
INTEGER, BIGINT, SMALLINT
DECIMAL(10,2), NUMERIC(10,2)
REAL, DOUBLE PRECISION

# Character types
CHAR(n), VARCHAR(n), TEXT

# Date/Time types
DATE, TIME, TIMESTAMP
TIMESTAMPTZ (with timezone)

# Boolean and others
BOOLEAN
JSON, JSONB
UUID
ARRAY (e.g., INTEGER[])
```

Constraints

Enforce data integrity rules at the table level.

```
# Primary key
id SERIAL PRIMARY KEY

# Foreign key
user_id INTEGER REFERENCES users(id)

# Unique constraint
email VARCHAR(100) UNIQUE

# Check constraint
age INTEGER CHECK (age >= 0)

# Not null
name VARCHAR(50) NOT NULL
```

Indexes: `CREATE INDEX`

Improve query performance with database indexes.

```
# Basic index
CREATE INDEX idx_username ON users(username);

# Unique index
CREATE UNIQUE INDEX idx_unique_email
   ON users(email);

# Composite index
CREATE INDEX idx_user_date
   ON orders(user_id, created_at);

# Partial index
CREATE INDEX idx_active_users
   ON users(username) WHERE active = true;

# Drop index
DROP INDEX IF EXISTS idx_username;
```

Sequences: `CREATE SEQUENCE`

Generate unique numeric values automatically.

```
# Create sequence
CREATE SEQUENCE user_id_seq;

# Use sequence in table
CREATE TABLE users (
   id INTEGER DEFAULT nextval('user_id_seq'),
   username VARCHAR(50)
);

# Reset sequence
ALTER SEQUENCE user_id_seq RESTART WITH 1000;
```

CRUD Operations

Insert Data: `INSERT`

Add new records to database tables.

```
# Insert single record
INSERT INTO users (username, email)
VALUES ('john_doe', 'john@example.com');

# Insert multiple records
INSERT INTO users (username, email) VALUES
   ('alice', 'alice@example.com'),
   ('bob', 'bob@example.com');

# Insert with returning
INSERT INTO users (username, email)
VALUES ('jane', 'jane@example.com')
RETURNING id, created_at;

# Insert from select
INSERT INTO archive_users
SELECT * FROM users WHERE active = false;
```

Update Data: `UPDATE`

Modify existing records in database tables.

```
# Update specific records
UPDATE users
SET email = 'newemail@example.com'
WHERE username = 'john_doe';

# Update multiple columns
UPDATE users
SET email = 'new@example.com',
   updated_at = NOW()
WHERE id = 1;

# Update with subquery
UPDATE orders
SET total = (SELECT SUM(price) FROM order_items
         WHERE order_id = orders.id);
```

Select Data: `SELECT`

Query and retrieve data from database tables.

```
# Basic select
SELECT * FROM users;

# Select specific columns
SELECT id, username, email FROM users;

# Select with conditions
SELECT * FROM users
WHERE active = true AND created_at > '2024-01-01';

# Select with ordering and limits
SELECT * FROM users
ORDER BY created_at DESC
LIMIT 10 OFFSET 20;
```

Delete Data: `DELETE`

Remove records from database tables.

```
# Delete specific records
DELETE FROM users
WHERE active = false;

# Delete with subquery
DELETE FROM orders
WHERE user_id IN (
   SELECT id FROM users WHERE active = false
);

# Delete all records
DELETE FROM temp_table;

# Delete with returning
DELETE FROM users
WHERE id = 5
RETURNING *;
```

Advanced Querying

Joins: `INNER/LEFT/RIGHT JOIN`

Combine data from multiple tables using various join types.

```
# Inner join
SELECT u.username, o.total
FROM users u
INNER JOIN orders o ON u.id = o.user_id;

# Left join
SELECT u.username, o.total
FROM users u
LEFT JOIN orders o ON u.id = o.user_id;

# Multiple joins
SELECT u.username, o.total, p.name
FROM users u
JOIN orders o ON u.id = o.user_id
JOIN products p ON o.product_id = p.id;
```

Subqueries & CTEs

Use nested queries and common table expressions for complex operations.

```
# Subquery in WHERE
SELECT * FROM users
WHERE id IN (SELECT user_id FROM orders);

# Common Table Expression (CTE)
WITH active_users AS (
   SELECT * FROM users WHERE active = true
)
SELECT au.username, COUNT(o.id) as order_count
FROM active_users au
LEFT JOIN orders o ON au.id = o.user_id
GROUP BY au.username;
```

Aggregation: `GROUP BY`

Group data and apply aggregate functions for analysis.

```
# Basic grouping
SELECT status, COUNT(*) as count
FROM orders
GROUP BY status;

# Multiple aggregations
SELECT user_id,
      COUNT(*) as order_count,
      SUM(total) as total_spent,
      AVG(total) as avg_order
FROM orders
GROUP BY user_id
HAVING COUNT(*) > 5;
```

Window Functions

Perform calculations across related rows without grouping.

```
# Row numbering
SELECT username, email,
      ROW_NUMBER() OVER (ORDER BY created_at) as
row_num
FROM users;

# Running totals
SELECT date, amount,
      SUM(amount) OVER (ORDER BY date) as
running_total
FROM sales;

# Ranking
SELECT username, score,
      RANK() OVER (ORDER BY score DESC) as rank
FROM user_scores;
```

Data Import & Export

Transfer data between PostgreSQL and external files or systems.

CSV Import: `COPY`

Import data from CSV files into PostgreSQL tables.

```
# Import from CSV file
COPY users(username, email, age)
FROM '/path/to/users.csv'
DELIMITER ',' CSV HEADER;

# Import with specific options
COPY products
FROM '/path/to/products.csv'
WITH (FORMAT csv, HEADER true, DELIMITER ';');

# Import from stdin
\copy users(username, email) FROM STDIN WITH CSV;
```

CSV Export: `COPY TO`

Export PostgreSQL data to CSV files.

```
# Export to CSV file
COPY users TO '/path/to/users_export.csv'
WITH (FORMAT csv, HEADER true);

# Export query results
COPY (SELECT username, email FROM users WHERE
active = true)
TO '/path/to/active_users.csv' CSV HEADER;

# Export to stdout
\copy (SELECT * FROM orders) TO STDOUT WITH CSV
HEADER;
```

Backup & Restore: `pg_dump`

Create database backups and restore from backup files.

```
# Dump entire database
pg_dump -U username -h hostname database_name >
backup.sql

# Dump specific table
pg_dump -U username -t table_name database_name >
table_backup.sql

# Compressed backup
pg_dump -U username -Fc database_name >
backup.dump

# Restore from backup
psql -U username -d database_name < backup.sql

# Restore compressed backup
pg_restore -U username -d database_name
backup.dump
```

JSON Data Operations

Work with JSON and JSONB data types for semi-structured data.

```
# Insert JSON data
INSERT INTO products (name, metadata)
VALUES ('Laptop', '{"brand": "Dell", "price": 999.99}');

# Query JSON fields
SELECT name, metadata->>'brand' as brand
FROM products
WHERE metadata->>'price'::numeric > 500;

# JSON array operations
SELECT name FROM products
WHERE metadata->'features' ? 'wireless';
```

User Management & Security

Create and manage database users, roles, and permissions.

Create Users & Roles

Manage database access with users and roles.

```
# Create user
CREATE USER myuser WITH PASSWORD 'secretpassword';

# Create role
CREATE ROLE readonly_user;

# Create user with specific privileges
CREATE USER admin_user WITH
   CREATEDB CREATEROLE PASSWORD 'adminpass';

# Grant role to user
GRANT readonly_user TO myuser;
```

Permissions: `GRANT/REVOKE`

Control access to database objects through permissions.

```
# Grant table permissions
GRANT SELECT, INSERT ON users TO myuser;

# Grant all privileges on table
GRANT ALL ON orders TO admin_user;

# Grant database permissions
GRANT CONNECT ON DATABASE mydb TO myuser;

# Revoke permissions
REVOKE INSERT ON users FROM myuser;
```

View User Information

Check existing users and their permissions.

```
# List all users
\du

# View table permissions
SELECT table_name, privilege_type, grantee
FROM information_schema.table_privileges
WHERE table_schema = 'public';

# Check current user
SELECT current_user;

# View role memberships
SELECT r.rolname, r.rolsuper, r.rolcreaterole
FROM pg_roles r;
```

Password & Security

Manage user passwords and security settings.

```
# Change user password
ALTER USER myuser PASSWORD 'newpassword';

# Set password expiration
ALTER USER myuser VALID UNTIL '2025-12-31';

# Create user without login
CREATE ROLE reporting_role NOLOGIN;

# Enable/disable user
ALTER USER myuser WITH NOLOGIN;
ALTER USER myuser WITH LOGIN;
```

Performance & Monitoring

Query Analysis: `EXPLAIN`

Analyze query execution plans and optimize performance.

```
# Show query execution plan
EXPLAIN SELECT * FROM users WHERE active = true;

# Analyze with actual execution stats
EXPLAIN ANALYZE
SELECT u.username, COUNT(o.id)
FROM users u
LEFT JOIN orders o ON u.id = o.user_id
GROUP BY u.username;

# Detailed execution information
EXPLAIN (ANALYZE, BUFFERS, VERBOSE)
SELECT * FROM large_table WHERE indexed_col = 'value';
```

Database Maintenance: `VACUUM`

Maintain database performance through regular cleanup operations.

```
# Basic vacuum
VACUUM users;

# Full vacuum with analyze
VACUUM FULL ANALYZE users;

# Auto-vacuum status
SELECT schemaname, tablename, last_vacuum,
last_autovacuum
FROM pg_stat_user_tables;

# Reindex table
REINDEX TABLE users;
```

Monitoring Queries

Track database activity and identify performance issues.

```
# Current activity
SELECT pid, usename, query, state
FROM pg_stat_activity
WHERE state != 'idle';

# Long running queries
SELECT pid, now() - query_start as duration, query
FROM pg_stat_activity
WHERE state != 'idle'
ORDER BY duration DESC;

# Kill specific query
SELECT pg_terminate_backend(pid) WHERE pid = 12345;
```

Database Statistics

Get insights into database usage and performance metrics.

```
# Table statistics
SELECT schemaname, tablename, n_tup_ins, n_tup_upd,
n_tup_del
FROM pg_stat_user_tables;

# Index usage statistics
SELECT schemaname, tablename, indexname,
idx_tup_read, idx_tup_fetch
FROM pg_stat_user_indexes;

# Database size
SELECT pg_size_pretty(pg_database_size('mydatabase'));
```

Advanced Features

Leverage PostgreSQL's advanced capabilities for complex database operations.

Views: `CREATE VIEW`

Create virtual tables to simplify complex queries and provide data abstraction.

```
# Create simple view
CREATE VIEW active_users AS
SELECT id, username, email
FROM users WHERE active = true;

# Create view with joins
CREATE OR REPLACE VIEW order_summary AS
SELECT u.username, COUNT(o.id) as total_orders,
      SUM(o.total) as total_spent
FROM users u
LEFT JOIN orders o ON u.id = o.user_id
GROUP BY u.id, u.username;

# Drop view
DROP VIEW IF EXISTS order_summary;
```

Triggers & Functions

Automate database operations with stored procedures and triggers.

```
# Create function
CREATE OR REPLACE FUNCTION update_timestamp()
RETURNS TRIGGER AS $$
BEGIN
   NEW.updated_at = NOW();
   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

# Create trigger
CREATE TRIGGER update_user_timestamp
   BEFORE UPDATE ON users
   FOR EACH ROW
   EXECUTE FUNCTION update_timestamp();
```

Transactions

Ensure data consistency with transaction control.

```
# Begin transaction
BEGIN;

UPDATE accounts SET balance = balance - 100
WHERE id = 1;

UPDATE accounts SET balance = balance + 100
WHERE id = 2;

# Commit transaction
COMMIT;

# Rollback if needed
ROLLBACK;

# Savepoints
SAVEPOINT my_savepoint;
ROLLBACK TO my_savepoint;
```

Configuration & Tuning

Optimize PostgreSQL server settings for better performance.

```
# View current configuration
SHOW shared_buffers;
SHOW max_connections;

# Set configuration parameters
SET work_mem = '256MB';
SET random_page_cost = 1.1;

# Reload configuration
SELECT pg_reload_conf();

# Show configuration file location
SHOW config_file;
```

psql Configuration & Tips

Customize your PostgreSQL command-line experience for maximum productivity.

Connection Files: `.pgpass`

Store database credentials securely for automatic authentication.

```
# Create .pgpass file (format:
hostname:port:database:userna
me:password)
echo
"localhost:5432:mydatabase:my
user:mypassword" >> ~/.pgpass

# Set proper permissions
chmod 600 ~/.pgpass

# Use connection service file
# ~/.pg_service.conf
[mydb]
host=localhost
port=5432
dbname=mydatabase
user=myuser
```

psql Configuration: `.psqlrc`

Customize psql startup settings and behavior.

```
# Create ~/.psqlrc file with
custom settings
\set QUIET on
\timing on
\set PROMPT1 '%n@%M:%>
%`date` %R%# '
\set HISTSIZE 5000
\set COMP_KEYWORD_CASE
upper
\x auto
\set QUIET off

# Custom aliases
\set show_slow_queries 'SELECT
query, mean_time, calls FROM
pg_stat_statements ORDER BY
mean_time DESC LIMIT 10;'
```

Environment Variables

Set PostgreSQL environment variables for easier connections.

```
# Set in your shell profile
export PGHOST=localhost
export PGPORT=5432
export
PGDATABASE=mydatabase
export PGUSER=myuser

# Then simply connect with
psql

# Or use specific environment
PGDATABASE=testdb psql
```

Common psql Meta-Commands

Essential psql backslash commands for database navigation and information.

Database Information

Get information about database objects and structure.

```
# List databases
\l, \l+

# List tables in current database
\dt, \dt+

# List views
\dv, \dv+

# List indexes
\di, \di+

# List functions
\df, \df+

# List sequences
\ds, \ds+
```

Table Information

Detailed information about specific tables and their structure.

```
# Describe table structure
\d table_name
\d+ table_name

# List table constraints
\d+ table_name

# Show table permissions
\dp table_name
\z table_name

# List foreign keys
SELECT * FROM information_schema.table_constraints
WHERE constraint_type = 'FOREIGN KEY';
```

Output & Formatting

Control how psql displays query results and output.

```
# Toggle expanded output
\x

# Change output format
\H  -- HTML output
\t  -- Tuples only (no headers)

# Output to file
\o filename.txt
SELECT * FROM users;
\o  -- Stop output to file

# Execute SQL from file
\i script.sql

# Edit query in external editor
\e
```

Timing & History

Track query performance and manage command history.

```
# Toggle timing display
\timing

# Show command history
\s

# Save command history to file
\s filename.txt

# Clear screen
\! clear  -- Linux/Mac
\! cls   -- Windows

# Show last error
\errverbose
```

Reference: This cheatsheet covers essential PostgreSQL commands and modern practices for efficient database management, SQL development, and administration workflows.