PostgreSQL (Postgres)

💡 What is PostgreSQL?

PostgreSQL (also called **Postgres**) is a **powerful, open-source object-relational database system (ORDBMS)**. It's known for its reliability, robustness, and advanced features.

- ✅ **Open-source**: Completely free to use under the PostgreSQL License (similar to MIT).
- 🛠️ **Object-relational**: Supports traditional relational database features, plus modern object-oriented features.
- 🔧 **Extensible**: You can define custom data types, operators, and even write functions in different languages (like Python, JavaScript, or SQL itself).

🧱 Core Architecture

PostgreSQL uses a **client-server model** with a **multi-process architecture**:

🔗 Client-Server Model

- **Clients** send SQL queries to the server.
- **Server** processes queries and sends results back.

🧩 Key Components

1. **Postmaster**: The main daemon that handles startup and shutdown.
2. **Backends**: Each client connection spawns its own server process.
3. **Shared Buffers**: A shared memory area for caching data.
4. **WAL (Write-Ahead Logging)**: Ensures durability—logs changes before they're written to disk.
5. **Background processes**: Autovacuum, WAL writer, checkpointer, etc.

🔑 Key Features

🔍 ACID Compliance

- **Atomicity**, **Consistency**, **Isolation**, **Durability**
- Ensures reliability even in power failures or crashes.

🧠 Advanced SQL Features

- Window functions
- Common Table Expressions (CTEs)
- Full-text search
- JSON & JSONB support

🗃️ Data Types

- Standard: `int`, `text`, `date`, etc.
- Advanced: `json`, `uuid`, `array`, `hstore`, `xml`, `geometric types`, `money`, etc.

⚙️ Indexing Options

- B-Tree (default)
- Hash
- GiST (Generalized Search Tree)
- GIN (Generalized Inverted Index) – great for full-text search
- BRIN – efficient for large tables with natural ordering

🚀 Performance Features

- Query planner and optimizer
- Parallel queries
- Table partitioning
- Caching
- Materialized views

🔐 Security

- Role-based authentication
- SSL/TLS for encryption
- Row-level security

🛠️ Extensibility

- Write your own functions (PL/pgSQL, PL/Python, PL/Perl, etc.)
- Add new data types, operators
- Install extensions (e.g., `PostGIS`, `pg_stat_statements`, `uuid-ossp`)

📦 Popular Extensions

Extension	Use Case
PostGIS	Geospatial queries (GIS support)
pg_stat_statements	Query performance tracking
uuid-ossp	UUID generation
citext	Case-insensitive text type
tablefunc	Pivot tables and crosstab views

🛠️ Tools & Interfaces

- **psql** – Command-line client
- **pgAdmin** – GUI tool for managing PostgreSQL
- **DBeaver / DataGrip** – Third-party DB tools
- **JDBC/ODBC drivers** – Integration with other applications
- **ORMs** – SQLAlchemy, Django ORM, Sequelize, etc.

🌐 Use Cases

- Web applications (e.g., with Django, Rails)
- Business analytics and reporting
- Geographic Information Systems (GIS)
- IoT and time-series data
- Financial systems
- Data warehousing

📊 Comparison with Other Databases

Feature	PostgreSQL	MySQL	SQLite	Oracle	MongoDB
ACID	✅	✅	✅ (limited)	✅	❌
JSON Support	✅ Advanced	✅ Basic	✅	✅	✅ Native

Extensibility	✅ High	❌	❌	Limited	❌
SQL Standard Compliance	✅ High	Moderate	Moderate	High	❌
Open-source	✅	✅	✅	❌	✅
Performance	✅ Excellent	✅ Good	✅ Lightweight	✅ Enterprise-grade	✅ High for NoSQL

🔧 Installation

PostgreSQL runs on Linux, Windows, and macOS.

Example (Ubuntu):

```
sudo apt update
sudo apt install postgresql postgresql-contrib
```

Start PostgreSQL:

```
sudo service postgresql start
```

🔄 Common SQL Commands

```
-- Create a new database
CREATE DATABASE mydb;

-- Create a table
CREATE TABLE users (
      id SERIAL PRIMARY KEY,
   name TEXT NOT NULL,
   email TEXT UNIQUE NOT NULL
);

-- Insert data and data should always be in single quotes
INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');

-- Query data
SELECT * FROM users;
```

```
-- Update
UPDATE users SET name = 'Alicia' WHERE id = 1;

-- Delete
DELETE FROM users WHERE id = 1;

-- Show All Databases
\l or \list

-- Show All Tables in a database
\dt

-- See All Tables across all schemas
\dt *.*

-- To select Database or change database connection
\c database_name

-- To list all users
\du

-- List all schemas
\dn

-- List tables in a schema
\dt sales.*

-- To see current database name
select current_database();

-- To create Scehma
create schema if not exists schema_name;

-- To view all the Scehmas
SELECT schema_name
FROM information_schema.schemata;

-- To view current schema
Select current_schema();

-- To sekect or switch the scehma
```

```
SET search_path TO your_schema_name;

-- To see all view tables
\dv

--To describe the tables
\d table_name
```

Data Migration in Postgres using “pg_dump”

Data Migration for Schemas

To migrate data from one database to another database which is of same database type which is postgres then we use “pg_dump” to migrate data with table structure and schema also

First we need to select source and target databases and then we will use

```
sudo -u username pg_dump -d source_db -n source_schema -t source_schema.source_table
```

Then check the file is created or not using below command:

```
ls -lh /tmp/table_dump.sql
```

Copy it and move to the home folder

```
 cp /tmp/table_dump.sql ~/table_dump.sql
```

Then import the file in target database to create the sructure and add the data

```
sudo -u postgres psql -d target_db -f /tmp/table_dump.sql
```

This will give an error of schema not found and so that we first manually needs to create schema in that database

```
sudo -u postgres psql -d target_database -c "CREATE SCHEMA source_schema;"
```

Then use above command for migrating the table

```
sudo -u postgres psql -d target_db -f /tmp/table_dump.sql
```

Data Migration for Database

In this now we will migrate whole database using pg_dump we will create a file containing all the info about database

```
sudo -u postgres pg_dump -d migrated_db -f /tmp/full_db.sql
```

Check weather the file is created or not:

```
ls -lh /tmp/full_db.sql
```

Copy the file into tmp file

```
sudo cp ~/full_db.sql /tmp/full_db.sql
```

Finally run this command to migrate the data:

```
sudo -u postgres psql -d migrated_db -f /tmp/full_db.sql
```

Difference between pg_dump and psql

🧰 Tools Overview

Tool	Purpose	Direction
`pg_dump`	**Exports** data	Creates `.sql` dump file
`psql`	**Imports** data	Reads `.sql` into DB

🔄 Workflow: What's Happening?

🟡 Step 1: Export with `pg_dump`

```
pg_dump -U user -d db1 -f full_db.sql
```

This **generates a plain-text `.sql` file** that contains all the SQL commands needed to recreate the database (schemas, tables, inserts, constraints, etc.).

📦 Think of it as "backing up" or "extracting" the database into a script.

🟢 Step 2: Import with `psql`

```
psql -U user -d db2 -f full_db.sql
```

This tells PostgreSQL to **execute** all the SQL in `full_db.sql`, line by line, inside the `db2` database.

📩 Think of it as “restoring” or “loading” the data into a new DB.

🚫 Why You Can't Use `pg_dump` to Import

Because:

- `pg_dump` **writes data out**, it does **not** write it *into* a database.
- It’s a **read-only** tool — you can’t use it to "put" data back in.

✅ Summary

Task	Tool to Use	Command Example
Export a database	`pg_dump`	`pg_dump -U user -d db1 -f full_db.sql`
Import a dump	`psql`	`psql -U user -d db2 -f full_db.sql`