DBS – Week 1: Getting Started with PostgreSQL

Introduction

Welcome to the first lesson of our Database Systems course! Today, you will gain hands-on experience with PostgreSQL. In this lesson, you will:

1. Install PostgreSQL 17.X (all assignments will be evaluated using this major version).
2. Choose a database client and connect to your local PostgreSQL server.
3. Restore an existing database.
4. Perform some sample queries on the restored database.

By the end of this lesson, you will have installed PostgreSQL 17, set up your preferred SQL client, restored the Pagila sample database, and executed basic SQL queries. Let's embark on your journey to mastering PostgreSQL!

Environment Setup

Installing PostgreSQL

Before you can start working with SQL queries and sample databases, you need to install a PostgreSQL server. Unlike client-only tools (such as DBeaver, pgAdmin, or DataGrip), the database server is the engine that stores, manages, and retrieves your data. It listens for incoming connections—by default on TCP port **5432**—and processes SQL commands from your client applications.

Why Install a Database Server?

- Centralized Data Management: The server stores all your data in one place, making it easier to manage, backup, and secure.
- Concurrent Access: Multiple users and applications can connect and work with the data simultaneously.

- Network Accessibility: The server listens on a TCP port (default 5432) so that remote clients can connect, if needed.

PostgreSQL vs. Other Database Systems

MySQL is a powerful, server-based relational database management system. Both require a running server process and listen on network ports for client connections.

SQLite is an embedded database that doesn't require a separate server process. The entire database is stored in a single file and is best suited for lightweight, local applications. In contrast, PostgreSQL is designed for more robust, multi-user environments and supports advanced functionalities like transactions, concurrency, and network-based access. For example – SQLite is for sure used in your mobile application to store local date / cache.

Instructions

Microsoft Windows

1. Download the official installer from EnterpriseDB
2. Run the installer and follow the 🤦‍♂️ steps
3. When prompted, set a password for the default *postgres* user (bro / sis, don't forget this password plz)
4. Verify installation by opening SQL Shell (psql) from the Start Menu. Connect as postgres user with password you have just specified. Then execute *SELECT version();*

macOS

There are plenty of options how to install PostgreSQL server on the macOS (you can do that using brew or official binaries from EnterpriseDB). We are used for the installation using the Postgres.app which we find the easiest one:

1. Download Postgres.app
2. Drag Postgres.app to your Applications folder
3. Open Postgres.app and Initialize your database server
4. Configure your $PATH to contain */Applications/Postgres.app/Contents/Versions/latest/bin* – this will make PostgrteSQL client binaries (such as psql, pg_dump or pg_restore) available from your shell

5. Check the install instructions to find out everything you need to know about your setup. Chill – it's super simple.
6. After you initialize and start server from the Postgres.app GUI you can verify the installation by running `/Applications/Postgres.app/Contents/Versions/latest/bin/psql -U postgres` which will open your psql console where you can execute `SELECT version();`.

Linux based systems (Ubuntu / Debian)

We will use vendor default PostgreSQL packages and apt package manager. If you are not using a Debian-based system you are a big boy / girl so you can make it on your own (using yum, dnf or pacman).

1. Execute `sudo apt update` to update yours's package repositories
2. Execute `sudo apt install postgresql postgresql-contrib` to install all required packages.
3. Execute `sudo systemctl enable postgresql.service --now` if you want to start the PostgreSQL automatically on the system boot. If you want to start your server manually you can do it by executing `sudo systemctl start|stop postgresql.service`
4. You can verify your installation by executing `sudo -i -u postgres` (which will log you in as system postgres user), then you can run `psql` (which will open your PostgreSQL console under postgres user) where you can execute `SELECT version();`.

Choosing SQL Client

After installing PostgreSQL, the next step is to choose a client to interact with your server. A PostgreSQL client provides a graphical or command-line interface for executing SQL queries, managing your databases, and performing administrative tasks. Here are a few popular options.

DBeaver

DBeaver is a free, open-source, multi-platform database tool that supports PostgreSQL and many other database systems. It offers an intuitive interface for browsing data, executing queries, and visualizing database schemas.

pgAdmin

pgAdmin is popular graphical management tool for PostgreSQL. It provides a web-based interface for managing your database, running queries, and viewing detailed database structures and logs.

DataGrip

DataGrip, developed by JetBrains, is a powerful, cross-platform database IDE that supports PostgreSQL along with a wide range of other databases. It offers advanced features such as intelligent code completion, on-the-fly analysis, and version control integration. Note that DataGrip is a commercial product with a free license for education.

Restoring database

In this section, we'll walk through restoring the Pagila sample database—a popular example database designed for PostgreSQL—using both the command-line interface (psql) and pgAdmin. This exercise will help you understand how to load a database dump into your PostgreSQL server so you can begin practicing SQL queries.

Ensure that you have downloaded the Pagila dump file (e.g., `pagila.sql` or a custom-format dump such as `pagila.dump`) from Microsoft Teams. If you feel brave, feel free to try it on your own using the official Pagila GitHub repository (the process slightly differs from our customized dump, but the data remain the same).

You can choose from two different backup formats – custom backup and raw SQL. Play with both of them. Can you spot the difference? Discuss the aftermath of using different backup formats.

Using command line interface

You can communicate with the database server using CLI tools installed with PostgreSQL. Keep in mind that these tools must be available in your `$PATH` environment variable (this should be the case if you have followed the installation process correctly). The paths to these tools may differ depending on the operating system you use or the version/method you used to install PostgreSQL. These tools should also work on

Microsoft Windows, but please check the documentation before using them (this workbook was mainly tested on macOS and Linux environments).

First, you need to create a new database. In our example, we will create a database called **pagila**. You can do this by calling the following command:

```
createdb -U postgres pagila
```

Alternatively, you can execute the command using SQL with the **psql** tool:

```
psql -U postgres -c "CREATE DATABASE pagila;"
```

Once the database is created, you can proceed with restoring the schema and data.

If your backup is a plain SQL file, you can use **psql**. The command below will connect to the **pagila** database on localhost as the PostgreSQL user and execute the file:

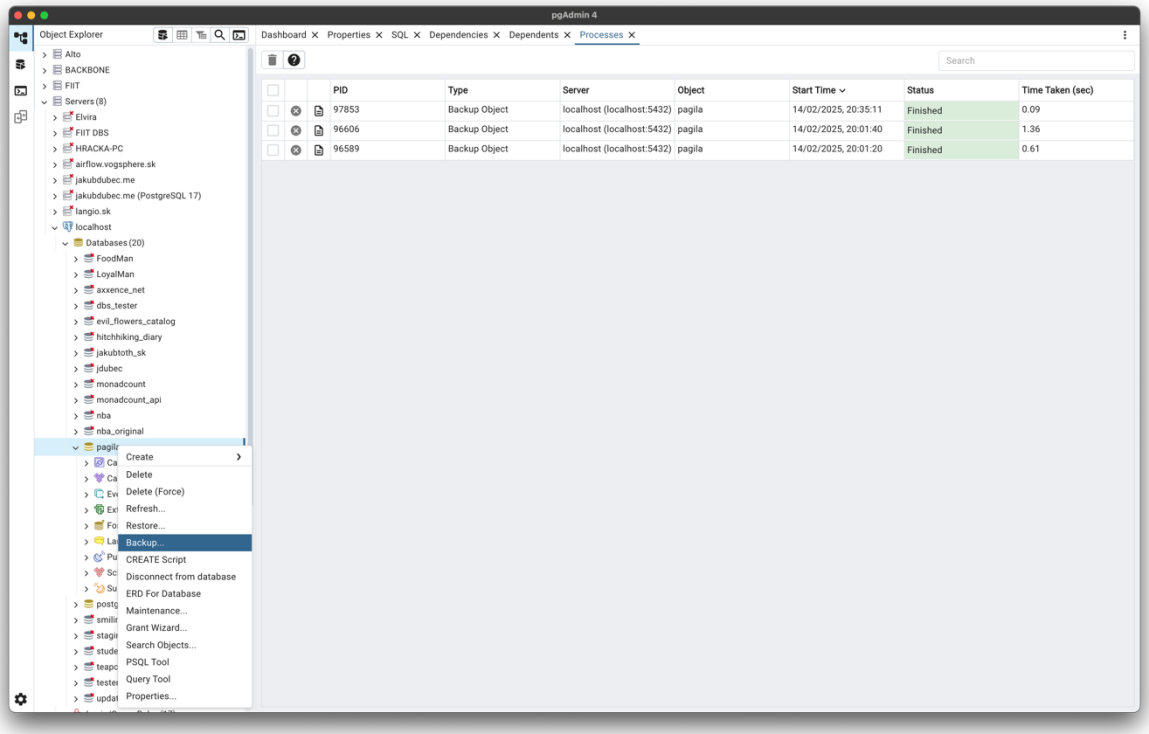
```
psql -U postgres -d pagila -f pagila.sql
```

If your backup is in a custom format, you can use **pg_restore**. Again, this command connects to localhost as the user **postgres** and restores to the **pagila** database:

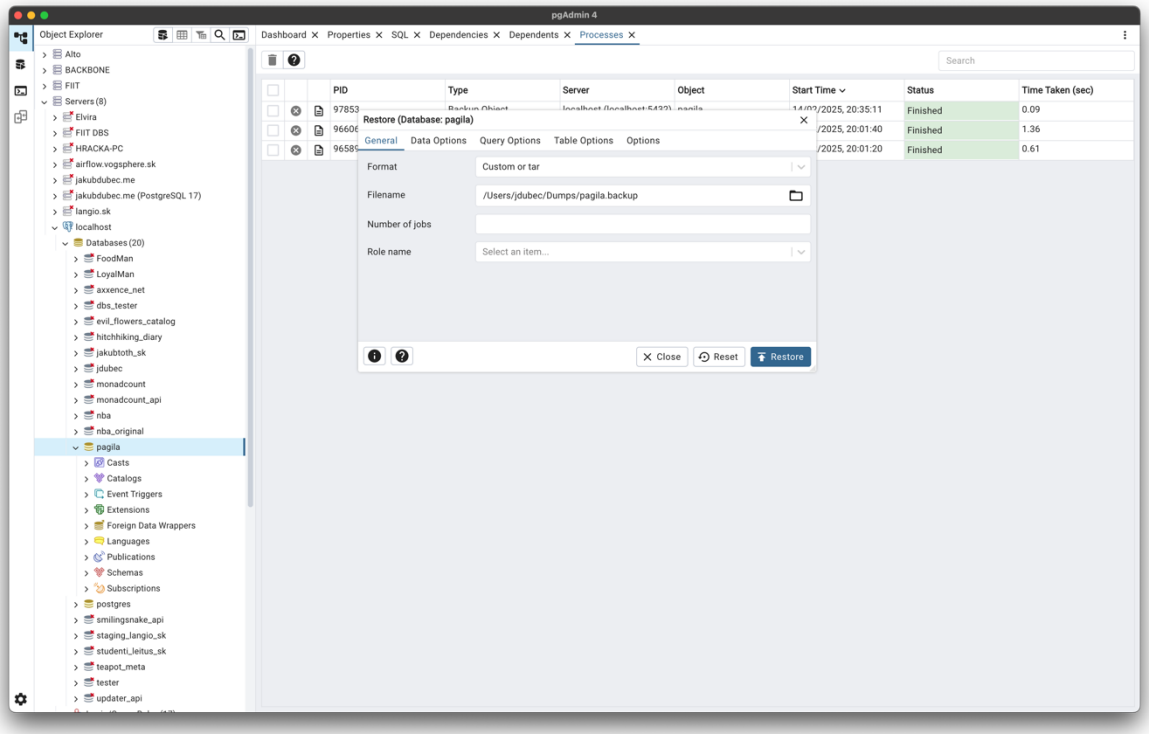
```
pg_restore -U postgres -d pagila pagila.backup
```

Using pgAdmin

If you're not a big fan of using CLI applications — or if you from the Windows universe — you can easily use pgAdmin for your database backup and restore process. In the image below, you can see where to find the Backup and Restore tools in pgAdmin. As you'll notice, the database should already be created (though this sometimes depends on the type of backup you have). Keep in mind that the entire restoration process is essentially a graphical front-end for the CLI tools (e.g., psql or pg_restore).

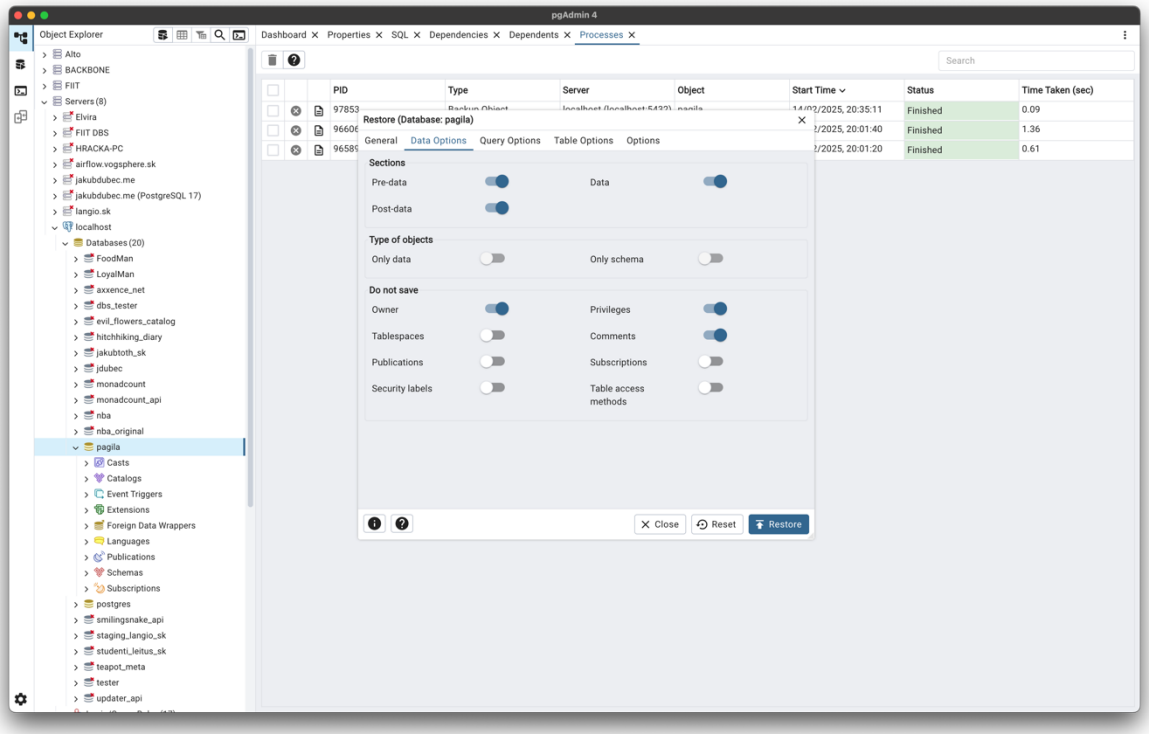


This method works for restoring backups in the custom format. If you have a plain-text backup, you should use the psql tool instead. Yes, there is a way to do it via the user interface, but beware: trying to open a huge backup file in the editor can make your application throw a tantrum—er, crash.



When migrating between different database servers with different users, it's always a good idea to skip importing the owner, privileges, or comments (this depends on which user is running the restoration process). In this example, the backup might fail if it references nonexistent users as owners or contains any comments in the backup (remember, there's a specific grant needed for creating comments in the database).

The configuration described here should be sufficient for most development processes.



Working with a Sample Database

In this chapter, we will explore the Pagila sample database. Pagila is a PostgreSQL adaptation of the well-known Sakila database and contains several tables related to films, actors, languages, and more. The queries below are designed for beginners who are just learning SQL. Each query is explained step by step.

Display a Few Actors

```
SELECT * FROM actor LIMIT 5;
```

This query selects all columns from the `actor` table but only shows the first 5 rows. This is a great way to quickly inspect the data in a table.

List Actor Names

```
SELECT actor_id, first_name, last_name FROM actor;
```

Here, we select only three columns (`actor_id`, `first_name`, and `last_name`) from the `actor` table. This helps you focus on the most important details: the unique identifier and the names of the actors.

Show Film Titles and Release Years

```
SELECT title, release_year FROM film LIMIT 10;
```

This query retrieves the `title` and `release_year` columns from the `film` table, displaying the first 10 films. It provides a simple view of the films available in the database.

Count the Number of Films

```
SELECT COUNT(*) AS total_films FROM film;
```

Using the `COUNT(*)` function, this query calculates the total number of rows (films) in the `film` table. The result is labeled as `total_films`.

List Films with Their Language

```
SELECT f.title, l.name AS language
FROM film AS f
JOIN language AS l ON f.language_id = l.language_id
LIMIT 10;
```

This query combines data from two tables:

- **film (aliased as f):** Contains film details.
- **language (aliased as l):** Contains language names.

The `JOIN` operation connects the two tables by matching `f.language_id` with `l.language_id`. The result displays the film title alongside the name of its language, and limits the output to 10 rows.