**PostGreSQL (aka Postgres)**

PostgreSQL (or simply postgress) is a free, open source implementation of a relational database management system (aka a “relational database”). It supports (uses) the SQL database language.

We will be

1. Installing postgres and the pgAdmin4 GUI tool
2. Adding several example databases to our installation of the postgres server, using pgAdmin4 to install existing databases
3. Learning how to use the command line to extract information from the database using SQL commands.

Later in the course, we will learn to connect Python to the database and use SQL and/or Python commands to pull data from the database so that we can analyze them in Python.

There is a lot more to database that we won’t cover in this course, such as how SQL databases are designed, how tables are created, how to add data to a table, what are views and triggers and a whole range of other database ideas. Creating databases is a course in and off itself, the goal of our work with postgres in this course is just to understand how to extract data from the database to make use of it. This is referred to as “pulling data”, or “querying data” or “extracting data”, and being able to do this well is a key skill for a data scientist.

We will get into extracting information from a database using the SELECT command, and how to join data from multiple tables using JOINS. You can do a lot of analysis just using SQL commands, or you can simply extract data and process it in R or Python. You will find that the SQL SELECT command (and related commands) are similar to the commands for sorting and slicing in the R Tidyverse package, or in the Python “Polars” library. This is not accidental.

**Relational Databases**

Relational databases make use of design principles developed by the mathematician Edgar Codd in the 1970s while working for IBM. Along with the relational database came the Structured Query Language (SQL). Relational databases use flat data tables (rows are observations or events, columns are varialbes) with keys that act as unique row identifiers and allow the tables to be related to one another. Codd developed an extensive set of rules that allow for robust and reliable storage and manipulation of data within the database.

**Database popularity, February 2024**

1. Oracle- commercial, relational database using SQL
2. MySQL- open source, relational database using SQL
3. Microsoft SQL Server-commercial, relational database using SQL
4. PostgreSQL-open source, supports SQL
5. MongoDB- NOSQL (not only SQL) database-document based DB, can be run in the cloud, SQL commands can be used within MongoDB
6. IBM Db2-commercial, relational database using SQL, offers NOSQL in form of JSON and XML
7. Redis- open source, NOSQL, uses key-value pairs
8. ElasticSearch- NOSQL, scales well, good for large data sets
9. Microsoft Access-GUI driven, has most of SQL available, best at smaller databases
10. SQLite-a derivative of MySQL, requires no server, can run within Python, runs on cell phones and other small devices- has most SQL abilities in a tiny package-open source

Note: Different sources have slightly different orderings to the rankings, and MariaDB and Firebase appear on some lists as well.

Relational databases use *flat data tables* and *keys* to connect records together. Most relational databases use SQL queries to extract information. There are a variety of different types of NOSQL databases, which allow for more complex, multi-dimensional (not flat) forms of data storage. Some of these are document-based systems, with open form documents in formats close to JSON or XML used to store data, others use key-value pairs much like a Python Dictionary. There is some overlap between these forms.

**Installing PostgreSQL**

1. <https://www.postgresql.org/download/>

Download the most recent version for your operating system

1. Install from the download folder- accept all the defaults in the install procedure
2. Set a password for the “superuser”- ie the database administrator, ie you. Write this down, don’t lose it.
3. Select the default port 5432

**Connecting to database servers**

There are several different ways to connect to a Database server, we will use all of these at least a little.

1. Through a command window. The simplest way to connect to a database is through a command window. If you learn to use the command window well, then you can easily adapt to the other approaches. The Command window for PostgreSQL is installed with the database, you can access it from the start menu on a windows computer.
2. Through a GUI database utility. Most databases have a GUI interface, kind of like an IDE. For a lot of utility tasks, like adding users, importing databases, etc, the GUI interfaces are often very fast and helpful, you don’t need to be as proficient as you do when using the command line. Most GUIs have a command window available in them, so you can use the command window from inside the GUI.

The GUI for PostgreSQL is called pgAdmin4 we will use it to install a couple of example databases to work with. The GUI for mySQL is called mySQL workbench, the GUI for sqlite is called SQLiteStudio. MySQL and sqlite are other commonly used, free, implementations of SQL databases.

1. From software packages- many tools, such as Tableau or PowerBI, can connect to a database, sometimes requiring a software driver called a database connector to connect to the database server. From inside the application, you can run SQL commands on the database to pull data into the application.
2. From languages like R or Python, you can use a library or a driver to connect to the database and then pull data from the database into the working space of the language (often storing it in a dataframe in R or Python). Some libraries have special commands for working with databases, other libraries simply pass SQL commands to the server and collect the results.

**Setting up a couple of example databases**

See the separate instructions for installing the bank, dvdrental and chinook databases into your installation of the postgres server. You will need to install the databases before proceeding with this set of instructions for how to look at those databases,

**Look at your databases in the SQL shell**

1. Start the SQL shell from the start menu under Postgress
2. The window will prompt “Server [localhost]:’- hit enter
3. The window will prompt “Database [postgres]” -hit enter
4. The window will prompt “Port [5432]:” hit enter
5. The window will prompt “Username [postgres]”- this is the default for the superuser, hit enter
6. The window will prompt “Password for user postgres”- type in the superuser password you entered when you set up postgress

**View the databases available in your server**

1. Type “\l” on the command prompt. You should see customerManagement and devdrental, plus a few others, including postgres
2. To chose a database to connect with and use, type \connect dvdrental or \connect customerManagement

**Viewing the Tables in your Database**

Once you have chosen a database using \connect {database name} as discussed above, we can look to see what tables are in the database using one of several options

1. The postgres command \dt
2. The general purpose SQL SELECT command

SELECT \* FROM pg\_catalog.pg\_tables

The SELECT command selects one or more columns from an SQL data table.

The \* is a wildcard meaning all columns

FROM indicates that we will be specifying the data table to pull the data out of

pg\_catalog.pg\_tables will go to a database called pg\_catalog which is part of postgress and then look up the pg\_tables table which is a table that store the list of available tables.

**Looking at the Content of a Table: SELECT**

In SQL we typically write SQL commands in all caps. It is not required, but it is a convention, so it is wise to follow it. In your workplace, it may be expected or required, so might as well get used to it. If you are reading SQL documentation or tutorials you will see this convention used and it does make easier to follow documentation.

In a SELECT command, at a minimum, we must specify the columns (variables) that we want to see, and the name of the table.

Suppose we want to see all the variables from the table “store” in the database dvdrental.

Working from the SQL Shell

1. Make sure dvdrental is selected, using \connect dvdrental
2. Use the command \dt to see a list of all the tables in the database
3. Let’s see with is in “store”, we want to see all rows and all columns, the command to do this is

SELECT \* FROM store;

Notice that SQL commands always have to end with a semicolon (;)

1. Let’s see what is in films

SELECT \* FROM film

This turns out to be a bit overwhelming, too much to see at once

Hit ctrl-C to stop the display

We can use the LIMIT{} command to limit the number of lines display and we can specify the columns to display, \* gives us all of them, we can request only specific lines

SELECT title,description,rating FROM film LIMIT 5