Local installation

Linux is the recommended operating system for database applications. It is possible to run a Linux OS in a virtual machine, if you use Windows or Mac, or use a Postgres container. To keep things simple, install Postgres on your system directly. VMs and container are useful in a real-world project or productive environment.

Linux

Install the available PostgreSQL and PostGIS version from the sources. On Ubuntu/Debian based systems you can do that in one command

```
sudo apt install postgresql postgresql-client postgis
```

Windows

Luckily, on Windows there is a enterprise-edition installer that can be used to install Postgres. Installing from source directly is more complicated and in case the installer does not work, it's recommended to run a Linux virtual machine. Download the installer bundle from https://www.postgresql.org/download/. Use a version `>= 10 && < 13`. Download the EDB installer, which also already includes PgAdmin4. Run the installer.

- be sure to select the `StackBuilder` option during installation.
- After the PostgreSQL installation, the installer will start the `StackBuilder`. Open the option `spatial extensions` and select `PostGIS`. Install that as well.

Mac

Follow the instructions for Windows, but download the correct bundle.

Remote installations

This is only reference documentation, you don't need the remote installation.

A local database server is nice for development, bu doesn't really help anything in real world projects. Then, you need to install the software on the server itself, or rent any kind of cloud database solution. For a remote installation of PostgreSQL, you first have to login to the remote machine via SSH and run the local installation for Linux OS.

There are a few additional steps necessary to allow incoming connections to the database server, which are disabled by default. There are two levels of security, the request to the database server and the authentication in the management system. Both are disabled for remote connections by default.

Steps

- The location of PostgreSQL's configuration files varies with Postgres's version and the used host system. You may need to search for files to find the exact location.
- On Ubuntu 19 and 20 the configuration files are located at: `/etc/postgresql/10/main`, where `10` will be your major version number.
- To allow incoming requests from other physical machines, open `postgresql.conf` and change the following line:

```
#listen_addresses = 'localhost'
```

to

```
listen_addresses = '*'
```

- To allow authentication, you need to open `pg_hba.conf` and **append** this line to the end of the file:

```
host  all   all       0.0.0.0/0     md5
```

You will need administrator rights on Windows or root privileges on Linux and Mac to edit this file.

- restart the PostgreSQL service (on Linux). One of the following commands will get the job done:

```
sudo service postgresql restart
sudo systemctl restart postgresql
```

On Windows, there is a PostgreSQL system icon and in the context menu you will find a restart option.

NOTE The shown changes of the configuration allows **anyone** to connect from **anywhere** to the PostgreSQL server. Thus, the only thing preventing others from accessing your Server are the **Postgres user role passwords**. Thus, use these settings only if absolutely necessary or during development and be more restrictive in production systems.

Usually, you will put some kind of REST middleware between incoming HTTP requests and database responses.

Development

There are cases when you need to run a remote PostgreSQL instance during development, because the productive system will be distributed on several machines. In these cases it can be a wise decision to run the PostgreSQL database in a virtual machine on your computer. Choose a server version (without Desktop) of your preferred OS, usually Ubuntu, Fedora or Debian. This has many advantages like:

- With many server providers, like GCP, AWS, AZURE or IONOS you can simply upload the VM to the server.
- You develop with the correct system in your stack
- VMs can be completely shut down with a single click, or autostarted whenever your System boots. You can easily adapt this to your needs.

Mandatory Software

QGis

Linux

Run:

```
sudo apt install qgis
```

If you have any other 3.X version installed, that is fine as well. The command above might not install the latest version and no GRASSgis and SAGA Gis support (which is not needed in this lecture, but recommended in case you use QGis as your primary GIS system). Installation instructions can be found online.

Windows

Visit qgis.org and download a LTS of QGis 3. Follow the instructions.

Mac

Installers for Mac High Sierra (10.13) can be found at: https://qgis.org/en/site/forusers/download.html.

Recommended Software

PgAdmin4

Python

If you have Python installed on your system, simply run:

```
pip install pgadmin4
```

This works on all OS.

Linux

Run:

```
sudo apt install pgadmin4
```

Windows

PgAdmin4 should have been installed along with your PostgreSQL installation. If not, use the Python installer or visit pgadmin.org and follow the instructions.

Mac

If PgAdmin was not installed along with PostgreSQL, it is highly recommended to use Python for installing PgAdmin4.

Anaconda

If you don't have it anyway, it is most helpful to manage the Python and R environments on your system using Anaconda. It is available for all OS. It basically bundles the whole environment into one folder and gives you the ability to *activate* or *deactivate* this environment. This way you can run an arbitrary amount of different Python and R versions simultaneously on your system and version conflicts won't steal your valuable time anymore. If you provide code for others on CRAN or PyPI, you should only do that, if you tested the code against different versions. Anaconda is a very easy way to do so. The latest download links can always be found on the Anaconda website: https://anaconda.com/products/individual

Anaconda vs. Miniconda

You will find two different installers on anaconda.com: Anaconda and Miniconda. Although there are more differences, you can think of Anaconda as a Miniconda with a predefined environment. You can use both the same way. Anaconda has kind of every useful Python package already installed, while Miniconda is smaller.

Installation instruction can also be found on anaconda.com. They look something similar to the commands shown below:

Linux

```
curl https://repo.anaconda.com/archive/Anaconda3-2020.07-Linux-x86_64.sh --output anaconda.s
bash anaconda.sh
```

Follow the instructions and be sure to add the Anaconda location to your path. The install script will ask you that.

Windows

```
curl https://repo.anaconda.com/archive/Anaconda3-2020.07-Windows-x86_64.exe --output anacon
anaconda.exe
```

Follow the instructions of the graphical installer.

Mac

Yeah, whatever you have to do.

Hydenv

There is a tool available, solely for helping you to download and install data for the lecture or start practicing some exercises. Hydenv should work cross-platform with no significant differences between the main OS (Linux, Windows, Mac). The recommended way to install it, is as a Python package, although you need Python first. Then, simply run

```
pip install hydenv
```

You may also find a compiled version for your OS in the Github repository: https://github.com/data-hydenv/hydenv-database. Then, you just call the downloaded executable, instead of the Python module.

Example: To verify, that it's working by asking the tool for the help text.

As Python module:

```
python -m hydenv --help
```

As Windows executable:

```
hydenv.exe --help
```

Then, whenever you read `python -m hydenv` in the notes, or see it in the videos, replace it by `hydenv.exe`. Keep in mind, that this is a command line interface and it has to be called from the command line. No double-clicking.

Hydenv CLI

The usage of the Hydenv CLI is **strongly recommended**.

This document is more like a cheat sheet, don't learn it by heart.

Please be aware, that the software was written during the shift of the lecture to an online event, to make your life easier. Thus, it was not yet tested in classroom situations. That means you can run into bugs and unclear or even missing documentation.

Don't panic!

Drop an Issue on Github and I will take care of it.

Call the ClI

Depending on your installation, you can either call the CLI as a Python module (Python installation) or an executable (downloaded). In both cases, you have to call the CLI from the command line (thus **C**ommand **L**ine **I**nterface). The difference is only how you have to call the CLI. For these lecture notes, we will use the Python module. Therefore, whenever you read something like this:

```
python -m hydenv --help
```

The executable on Windows will be run like:

```
C:/path/to/hydenv.exe --help
```

So, don't double click on the executable, that will run and instantly close the application again.

On Windows, you can open a command prompt with `[Win] + [R]`, or search for `cmd.exe`.

Create the database

The recommended method is using the CLI here. However, you can also install the database manually. But keep in mind that the cli will do more than just installing the database. In case you do not use it, you have to create tables and upload data on your own, which will drastically increase your workload. Therefore, install the CLI and get in touch if anything goes wrong.

Use PgAdmin

Creating the database is a more complicated step, as you need root privileges to create new database instances. In case you feel a bit comfortable with PgAdmin4, you can use it, as it gives you more control. This is the way how any new database (e.g. for other projects) is created. There are four steps that have to be taken:

1. Create a new database called `hydenv`.
2. Create the PostGIS extension in that database.
3. Create a new user called `hydenv` (or whatever you want to use)
4. Grant all permissions on the new database to the new user.

In PgAdmin4 you can do all these steps using the GUI, or with the SQL prompt. In both cases, you have to **connect** to the new database after **step 1**. In the SQL prompt run the following commands.

```sql
CREATE DATABASE hydenv;
```

And then connected to that database:

```sql
CREATE EXTENSION postgis;
CREATE ROLE hydenv WITH PASSWORD 'hydenv';
GRANT ALL PRIVILEGES ON DATABASE hydenv TO hydenv;
```

Use hydenv CLI

You can also use the CLI, although the command is a bit confusing. You need to pass a connection with root privileges and specify how the new user should be named at the same time (the CLI will create the new user):

```
python -m hydenv database install --user=hydenv --password=hydenv --db-name=hydenv --connect
```

Here, `<password>` is the root password you chose during the installation of postgres. With the three flags `--user=hydenv --password=hydenv --db-name=hydenv` you can specify the database name, user name and user password for the database used in this lecture.

Another advantage of using the CLI, is that you can skip the next point 'Saving connections' as that was already done by the CLI.

Saving connections

Whenever you run a command with the CLI, you need to specify the connection to the database using the `--connection` flag. That is cumbersome. Thus, the CLI can save the connection for you to the disk. If done so, it will load it automatically, whenever needed.

```
python -m database save --user=hydenv --password=hydenv --host=localhost --port=5432 --dbnam
```

Note: This will save the password into a text-file in your home directory.

Only use this in a trusted environment. Alternatively, you could omit the password and set it as an Environment variable on every startup of the command line tool.

Windows:

```
set POSTGRES_PASSWORD=hydenv
```

Linux:

```
export POSTGRES_PASSWORD=hydenv
```

Build database schema

Install the schema like:

```
python -m hydenv init
```

Add the `--clean` flag, to drop existing tables first and populate default lookup values

```
python -m hydenv init --clean
```

On first run, you should add the `--clean` flag, to have all lookup tables filled.

Execute any command

To verify that installation and initialization were fine so far, you can use the `execute` command to run any SQL query. With the following example, you can count one of the prefilled tables:

```
python -m hydenv database execute --sql="SELECT count(*) as amount FROM terms"
```

This should return something like: `(48,)`.

Upload sample data

Uploading data into the database is a multi-step process. First, you need to create metadata entries. Then upload the actual data and link it to the correct metadata. Finally, you have to decide how to react to integrity issues of your data.

The `hydenv` has you covered here. It allows low-level access to the database by writing records one at a time. On top of that you can script any upload logic you need, in your preferred scripting language. Alternatively, there are so-called *examples*, which do all of this in one step. To get started with exercises, you can load one of last years HOBO campaigns, that took place during lecture in Freiburg.

```
python -m hydenv exercises --help
```

To upload all data collected in Winterterm 2018/2019 you can run this command:

```
python -m hydenv exercises hobo --terms=WT18
```

It will look up the metadata in a collaborative google sheet and import that. Then, the actual data is downloaded from Github and imported as well.