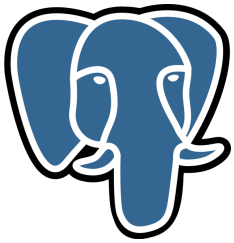
GIS – Tutorial 1 – Spatial Databases

University of Konstanz | Due Date: 13.11.2020 08:00

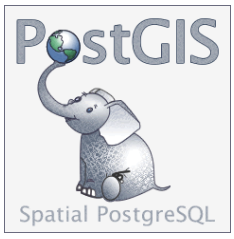
1 Submission Instructions

Please provide **one** well-formatted **PDF** file with all your submissions on schedule (i.e. before the deadline). Otherwise your submission will not be graded!

2 Background



PostgreSQL, also known as Postgres, is a free and open-source relational database management system (RDBMS) emphasizing extensibility and technical standards compliance.
For more information visit: https://www.postgresql.org/



PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL.
For more information visit: https://postgis.net/

2.1 Installation

This tutorial assumes that PostgreSQL and the PostGIS extension will be installed as a Docker container. If you want to manually install both of these programs please visit the following URLs:

- PostgreSQL: https://www.postgresql.org/download/
- PostGIS: https://postgis.net/install/

2.2 Database Tools

In general, it is helpful to use a database tool to quickly connect to the database and to try out queries. For that you can use your prefered database tool. For instance:

- DBeaver: https://dbeaver.io/
- DataGrip: https://www.jetbrains.com/de-de/datagrip/
- pgAdmin: https://www.pgadmin.org/

2.3 QGIS

If you want to quickly visualize some geographic data, QGIS is a free and open-source platform to quickly visualise and analyse geospatial information.

- QGis: https://www.qgis.org

2.4 Useful Docker commands for this assignment

During this tutorial we will often start & stop Docker containers, the following commands will be often used or might be helpful:

- *docker run [OPTIONS] IMAGE*
  https://docs.docker.com/engine/reference/run/
  Runs the *IMAGE* Docker container with the specified *[OPTIONS]*
- *docker ps*
  https://docs.docker.com/engine/reference/commandline/ps/
  Shows a list of currently running Docker containers
- *docker stop [OPTIONS] CONTAINER*
  https://docs.docker.com/engine/reference/run/
  Stops the *[CONTAINER]*. You can identify the container id or name using *docker ps*.
- *docker system prune -a*
  https://docs.docker.com/engine/reference/commandline/system_prune/
  Remove all unused containers, networks, images, etc.

3 Homework

Task 1: PostgreSQL + PostGIS Setup **4 Points**

1. Download the following Docker container that runs PostgreSQL + PostGIS

   docker pull kartoza/postgis:12.1

2. Ensure you can successfully run this Docker container by running the following command:

   docker run kartoza/postgis:12.1

   One of the last log outputs should be:

   database system is ready to accept connections

Submission: Output of the last 5 lines of the *docker run kartoza/postgis:12.1* command.

Task 2: Modifying the Docker container **4 Points**

By default the PostGIS Docker container has the user 'docker' with password 'docker' and creates a default database 'gis'. To change these values, we can set environment variables in the run command using the flag *-e*. However, when setting multiple variables the command can get quite large and complex. But we can also create our own Dockerfile which extends the PostGIS container.

1. Extend the stub below, set the following environment variables (*POSTGRES_USER, POSTGRES_PASS, POSTGRES_DBNAME*) and save it in a file named "Dockerfile":

```
# specify base image that we want to extend
FROM kartoza/postgis:12.1

# set environment variables that we want to change
...
```

   For more information, see: https://docs.docker.com/engine/reference/builder/#env

2. Check that you can successfully run this Docker container by running the two following commands:

   (a) Build the new Docker container and give it a name (tag):
   docker build - < Dockerfile -t my_own_postgis_container

   (b) Run the new Docker container and map the port 25432 to port 5432 in the container:
   docker run -p 25432:5432 my_own_postgis_container

   (c) Check that you can connect to the container on "localhost:25432" using your prefered tool:
   DBeaver, pgAdmin, QGis, DataGrip, ...
   Hint: Make sure you use the credentials specified in the environment variables above.

Submission: Output of the last 5 lines of the command *docker build - < Dockerfile -t my_own_postgis_container* and a list of all tables that were automatically created in your database in the schema public.

Task 3: Loading a dataset **4 Points**

After we've successfully created our PostGIS database we need to insert some data into it. An amazing data source is the OpenStreetMap project (https://www.openstreetmap.org) which provides data about roads, trails, cafés, railway stations, and much more, all over the world. For instance to get some information about Konstanz we can download the OpenStreetMap data of the governmental district of Freiburg (https://download.geofabrik.de/europe/germany/baden-wuerttemberg/freiburg-regbez-latest.osm.pbf). However this dataset is in the *Protocolbuffer Binary Format* and cannot be directly loaded into our database. For this we need another tool called *osm2pgsql* https://wiki.openstreetmap.org/wiki/Osm2pgsql.

1. Install *osm2pgsql* (Download binaries for Windows, use homebrew on Mac or install via package manager on Linux)

2. Import the *freiburg-regbez-latest.osm.pbf* dataset into the PostGIS database using the following command:
   osm2pgsql --database <database name> --host localhost --port 25432 --username <username> --password --create --slim --drop --latlong --hstore-all freiburg-regbez-latest.osm.pbf
   Hint: This may take a few minutes

3. After the command has run through, check that it successfully created the tables in your database containing the OpenStreetMap data. There should be 4 tables in total: *planet_osm_line*, *planet_osm_point*, *planet_osm_polygon*, and *planet_osm_roads*. You can again use your prefered tool to connect to the database *DBeaver, pgAdmin, QGis, DataGrip, ...*

Submission: Output of the last 5 lines of the command *osm2pgsql ....*

Task 4: Querying the database **8 Points**

To query the database we can now write SQL queries as you might know them from other relational database systems. As you might have guessed from the names of the tables created by *osm2pgsql* in the last task, *planet_osm_line* contains lines, *planet_osm_polygon* contains polygons, etc. To get, for instance, information about the area of the 'Landkreis Konstanz' you can issue the following query:

```
SELECT name, ST_Area(way::geography)/(1000*1000) as area_sqkm
FROM planet_osm_polygon
WHERE name = 'Landkreis Konstanz' AND admin_level = '6'
```

1. The query above returns 5 different polygons and their areas for the query regarding the area of the 'Landkreis Konstanz'. Usually you would only expect one result here. Discuss why you think it is the case, that the query returns 5 results instead of only one.
   Hint: It helps tremendously to visualize the results. You can, for instance, use QGIS to quickly visualize the different polygons.

2. Write a query that returns all information in the table *planet_osm_polygon* for **only** Konstanz Cathedral.
   Hint: The name of the cathedral is 'Münster Unserer Lieben Frau', but be careful, there is also a church in Radolfzell called 'Münster Unserer Lieben Frau'.

3. Calculate the distance between Konstanz Cathedral and the University of Konstanz.
   Hint: You can use the function ST_Distance for that. Check its documentation to find out how to get the distance in meters.

4. How many pubs and bars are there in the city of Konstanz?
   Hint: You can find the city of Konstanz with *name = 'Konstanz' AND admin_level = '8'*, also the table *planet_osm_points* contains more bars and pubs than *planet_osm_polygon.*

Submission: For subtask 1 please submit a short written discussion and for subtask 2-4 submit your SQL query as well as your result.