Storing a GeoNames Dump to Postgres

Ludwig Richter (ludwig.richter@posteo.de)

March 24, 2017

In this document, the creation and usage of a database infrastructure for storing a GeoNames dump in a PostgreSQL/PostGIS environment is described. The database schema is based on two documents provided by GeoNames[1,2] and a blog post[3]. For our purposes some small adjustments were made to improve its usability and efficiency. For the sake of simplicity, throughout this text a Linux based operating system is assumed as working environment.

1 Installation of PostgreSQL and PostGIS

Before creating a GeoNames database in PostGIS, you first need to install the two software packages PostgreSQL and PostGIS. You can download it from `http://www.postgresql.org/download/` and `http://postgis.net/install`, respectively. At the time of writing, PostgreSQL 9.5 and PostGIS 2.2.2 as wells as a GeoNames data dump downloaded on 7th of September 2016 were successfully used for this workflow. Newer versions should be usable without problems, though. You can also execute the following command on the command-line to install both programs:

```
sudo apt install sudo apt-get install postgresql-9.5-postgis-2.2
        pgadmin3 postgresql-contrib-9.5
```

2 Setting up the Database

Now, create a new database named e.g., `geonames`[4]. You can do this by using the command line and executing the following commands (assuming, you are using user `postgres`):

[1] `http://download.geonames.org/export/dump/readme.txt`

[2] `http://download.geonames.org/export/dump/countryInfo.txt`

[3] `http://jamescreel.net/blog/utilizing-the-geonames-dataset-in-postgres/`

[4] consider a single database for all PostgreSQL related data processing, in case you want table joins across different data sources (e.g. some data extracted from GeoNames, WikiData and OpenStreetMap). Joining across multiple databases is not possible!

```
createdb -U postgres geonames
psql -U postgres -d geonames -c 'CREATE EXTENSION postgis;'
```

As a convenient alternative, you can use pgAdmin[5]: create the database via the context menu and then execute the query `CREATE EXTENSION postgis;` on it in the query editor.

3 Setting up the Database Schema

After the basic setup it is time for creating the database schema. Considering the above quoted sources and the naming of the tables and attributes, the schema should be fully self-explanatory and is not described in detail here. The database schema is designed to be rather comprehensive and some tables like `admin1codes`, `admin2codes`, `languagecodes`, `featurecodes` or `continentinfo` may be omitted in the following workflow. They are only informative regarding the meaning of the different attribute values and the core database can be used without them. Note that in contrast to the documents quoted above, all `varchar` occurrences are defined as `text` datatype to avoid potential future conflicts caused by changing maximum string lengths[6]. Execute the following query e.g., in the PgAdmin query editor after having selected your new database as the current one:

```
CREATE TABLE geoname (
    geonameid int,
    name text,
    asciiname text,
    alternatenames text,
    latitude float,
    longitude float,
    fclass char(1),
    fcode text,
    country text,
    cc2 text,
    admin1 text,
    admin2 text,
    admin3 text,
    admin4 text,
    population bigint,
    elevation int,
    gtopo30 int,
    timezone text,
```

[5]You can download it from `http://www.pgadmin.org/download/` and it is strongly suggested to use this tool (usually, this tool is already included by default with the PostgreSQL installation)

[6]According to `http://stackoverflow.com/a/4849030` this is a valid approach and we have no need for strictly enforcing string length constraints since we do not edit the data or rely on its structure

```
    moddate date
);
CREATE TABLE alternatename (
    alternatenameid int,
    geonameid int,
    isolanguage text,
    alternatename text,
    ispreferredname boolean,
    isshortname boolean,
    iscolloquial boolean,
    ishistoric boolean
);
CREATE TABLE countryinfo (
    iso_alpha2 char(2),
    iso_alpha3 char(3),
    iso_numeric integer,
    fips_code text,
    name text,
    capital text,
    areainsqkm double precision,
    population integer,
    continent text,
    tld text,
    currencycode text,
    currencyname text,
    phone text,
    postalcodeformat text,
    postalcoderegex text,
    languages text,
    geonameid int,
    neighbors text,
    equivfipscode text
);
CREATE TABLE hierarchy (
    parentid int,
    childid int,
    type text
);
CREATE TABLE admin1codes (
    admin1 text,
    name text,
    asciiname text,
    geonameid int
);
CREATE TABLE admin2codes (
    admin2 text,
    name text,
    asciiname text,
    geonameid int
);
CREATE TABLE continentinfo (
    geonameid int,
    continent text
);
CREATE TABLE languagecodes (
    iso6393 text,
    iso6392 text,
    iso6391 text,
    languagename text
);
CREATE TABLE featurecodes (
    fclass char(1),
    fcode text,
    name text,
    description text
);
```

4 Download the GeoNames Dump

Download and unzip the current GeoNames data dump by executing the following commands on the command line:

```
wget http://download.geonames.org/export/dump/allCountries.zip
wget http://download.geonames.org/export/dump/alternateNames.zip
wget http://download.geonames.org/export/dump/countryInfo.txt
wget http://download.geonames.org/export/dump/admin1CodesASCII.txt
wget http://download.geonames.org/export/dump/admin2Codes.txt
wget http://download.geonames.org/export/dump/hierarchy.zip
wget http://download.geonames.org/export/dump/featureCodes_en.txt
unzip allCountries.zip
unzip alternateNames.zip
unzip hierarchy.zip
```

Important: Delete all header lines (starting with #) from the `countryInfo.txt` file, since the PostgreSQL `COPY` command is not capable of identifying them as header lines!

5 Insert the Dump to Postgres

Insert the dump into the database by issuing the following commands on the `psql` interface (you can open it either via the pgAdmin context menu Plugins / PSQL Console or by starting it from the command line with `psql --username=postgres -d geonames`), after having changed all occurrences of `your_dump_path` to the location of your GeoNames dump files (Note: no linebreaks are allowed within a command - we only use them to make this printing a bit more readable):

```
\COPY geoname (geonameid,name,asciiname,alternatenames,latitude,
   longitude,fclass,fcode,country,cc2,admin1,admin2,admin3,admin4,
   population,elevation,gtopo30,timezone,moddate) FROM 'your_dump_path/
   allCountries.txt' NULL AS '' ENCODING 'utf-8';
\COPY alternatename (alternatenameid,geonameid,isolanguage,alternatename
   , ispreferredname,isshortname,iscolloquial,ishistoric) FROM '
   your_dump_path/alternateNames.txt' NULL AS '' ENCODING 'utf-8';
\COPY countryinfo (iso_alpha2,iso_alpha3,iso_numeric,fips_code,name,
   capital,areainsqkm,population,continent,tld,currencycode,currencyname
   ,phone,postalcodeformat,postalcoderegex,languages,geonameid,neighbors
   ,equivfipscode) FROM 'your_dump_path/countryInfo.txt' NULL AS ''
   ENCODING 'utf-8';
\COPY hierarchy (parentid,childid,type) FROM 'your_dump_path/hierarchy.
   txt' NULL AS '' ENCODING 'utf-8';
\COPY admin1codes (admin1,name,asciiname,geonameid) FROM 'your_dump_path
   /admin1CodesASCII.txt' NULL AS '' ENCODING 'utf-8';
\COPY admin2codes (admin2,name,asciiname,geonameid) FROM 'your_dump_path
   /admin2Codes.txt' NULL AS '' ENCODING 'utf-8';
\COPY languagecodes (iso6393,iso6392,iso6391,languagename) FROM '
   your_dump_path/iso-languagecodes.txt' NULL AS '' ENCODING 'utf-8' CSV
    HEADER DELIMITER E'\t';
\COPY featurecodes (fcode,name,description) FROM 'your_dump_path/
   featureCodes_en.txt' NULL AS '' ENCODING 'utf-8' CSV DELIMITER E'\t';
```

Note: the use of `pqsl` and `\COPY` is suggested in order to avoid troubles with file access permissions (if `COPY` would be used a normal SQL statement, you would need to ensure to have the appropriate file access rights, which might be not possible depending on your working environment). Also `ENCODING 'utf-8'` is used to avoid problems caused by incorrectly guessed encodings.

6 Manually Adjust Incorrect or Missing Data

Manually fill the `continentinfo` table and correctly format the data in the `featurecodes` table (again, using e.g. the PgAdmin query editor):

```
INSERT INTO continentinfo(continent,geonameid)
VALUES ('AF',6255146),('AS',6255147),('EU',6255148),('NA',6255149), ('SA
   ',6255150),('OC',6255151),('AN',6255152);
DELETE FROM featurecodes WHERE fcode = 'null';
UPDATE featurecodes SET fclass = substring(fcode from 1 for 1);
UPDATE featurecodes SET fcode = substring(fcode from '[^\.]*$');
```

```
DELETE FROM hierarchy AS h WHERE NOT EXISTS (SELECT * FROM geoname WHERE
     childid = geonameid);
```

7 Add Primary and Foreign Key Constraints

Add the following primary and foreign key constraints to the database schema:

```
ALTER TABLE ONLY geoname
    ADD CONSTRAINT pk_geonameid PRIMARY KEY (geonameid);
ALTER TABLE ONLY alternatename
    ADD CONSTRAINT pk_alternatenameid PRIMARY KEY (alternatenameid),
    ADD CONSTRAINT fk_geonameid FOREIGN KEY (geonameid) REFERENCES
   geoname(geonameid);
ALTER TABLE ONLY countryinfo
    ADD CONSTRAINT pk_iso_alpha2 PRIMARY KEY (iso_alpha2),
    ADD CONSTRAINT fk_geonameid FOREIGN KEY (geonameid) REFERENCES
   geoname(geonameid);
ALTER TABLE ONLY admin1codes
    ADD CONSTRAINT pk_admin1code PRIMARY KEY (admin1),
    ADD CONSTRAINT fk_geonameid FOREIGN KEY (geonameid) REFERENCES
   geoname(geonameid);
ALTER TABLE ONLY admin2codes
    ADD CONSTRAINT pk_admin2code PRIMARY KEY (admin2),
    ADD CONSTRAINT fk_geonameid FOREIGN KEY (geonameid) REFERENCES
   geoname(geonameid);
ALTER TABLE ONLY hierarchy
    ADD CONSTRAINT fk_geonameid_parent FOREIGN KEY (parentid) REFERENCES
     geoname(geonameid),
    ADD CONSTRAINT fk_geonameid_child FOREIGN KEY (childid) REFERENCES
   geoname(geonameid);
ALTER TABLE ONLY continentinfo
    ADD CONSTRAINT pk_continent PRIMARY KEY (continent),
    ADD CONSTRAINT fk_geonameid FOREIGN KEY (geonameid) REFERENCES
   geoname(geonameid);
ALTER TABLE ONLY languagecodes
    ADD CONSTRAINT pk_languagename PRIMARY KEY (languagename);
ALTER TABLE ONLY featurecodes
    ADD CONSTRAINT pk_featurecode PRIMARY KEY (fclass,fcode);
```

8 Add Geometry Column

The following step is not required, if you create the database to use it for the `GeoNamesImporter` program of our gazetteer framework!

To enable the power of PostGIS functionality and to allow for conveniently using the data with external geographical information visualization tools such as QGIS[7], create a PostGIS geometry column. Then, insert the geometry data and create an index on it (this may take quite a while!):

```
ALTER TABLE geoname ADD COLUMN the_geom geometry(Point,4326);
UPDATE geoname SET the_geom = ST_PointFromText('POINT(' || longitude ||
   ' ' || latitude || ')', 4326);
CREATE INDEX idx_geoname_the_geom ON geoname USING gist(the_geom);
```

9 Add Extra Table for Full Admin Code

With the following script, you can add an extra table containing the full admin code per place as string. This table is required by the `GeoNamesImporter` program of our gazetteer framework!

```
    SELECT geonameid, COALESCE(country, '') || '.' || COALESCE(admin1,
   '') || '.' || COALESCE(admin2, '') || '.'  || COALESCE(admin3, '') ||
    '.'  || COALESCE(admin4, '') as admin4
    INTO admin4codes FROM geoname;

    ALTER TABLE admin4codes
      ADD CONSTRAINT pk_admin4_geonameid PRIMARY KEY (geonameid),
      ADD CONSTRAINT fk_geonameid FOREIGN KEY (geonameid) REFERENCES
   geoname (geonameid);

    CREATE INDEX admin4_idx ON admin4codes USING btree (admin4);
```

10 Establish Indices

Depending on your usage patterns, you might want to experiment with further indexes to increase the processing performance, especially with indexes on the name-columns and the administrative hierarchy. In the following, we present a few indexes that where useful:

```
-- the following indexes are only useful if you query names with lower()
   , which is often appropriate for place names
CREATE INDEX geoname_lower_idx ON geoname (lower(name));
CREATE INDEX altname_lower_idx on alternatename (lower(alternatename));
-- important index if you want to get aliases for a place name
   efficiently (we use this in GeoNamesImporter)
CREATE INDEX alternatename_geonamesid_idx ON alternatename USING btree(
   geonameid);
-- important index if you want to query links with a match-pattern  (we
   use this in GeoNamesImporter)
```

[7]http://www.qgis.org/en/site/

```
CREATE INDEX alternatename_link_pattern_idx ON alternatename USING btree
   (alternatename text_pattern_ops) WHERE isolanguage = 'link';
-- important indexes if you want to quickly gather hierarchical
   information
CREATE INDEX hp_idx on hierarchy (parentid);
CREATE INDEX hc_idx on hierarchy (childid);
-- useful indexes if you often operate on the fcode column (we use this
   in GeoNamesImporter)
CREATE INDEX fcode_idx ON geoname USING hash (fcode);
CREATE INDEX fclass_a_fcode_pattern_idx ON geoname(fcode
   text_pattern_ops) WHERE fclass = 'A';
CREATE INDEX fclass_p_fcode_pattern_idx ON geoname(fcode
   text_pattern_ops) WHERE fclass = 'P';
```

11 Run `VACUUM FULL`

It is strongly suggested to run `VACUUM FULL` on the geonames database, since the table sizes decrease considerably!

12 Final Remarks

As a final remark, the GeoNames documentation mentions that the field `alternativenames` in the table `geoname` is a short version of the `alternatenames` table. If you do not need to know the language of a name variant and all the related information, the field `alternativenames` will be enough and you can drop the `alternatenames` table. On the other hand, if you actually care about this information, you can drop the `alternativenames` field, instead. However, it was found that the number of aliases differ between the `alternatenames` table and the `alternativenames` field, so be aware that they are obviously not identical!