Designing a Relational Database for Spatial Querying using PostgresSQL with PostGIS

This tutorial is meant to provide an example of how to design a relational spatial database for spatial querying through the use of freely available resources. All you will need is a computer, an internet connection, and some patience.

In this exercise, you will learn how to use point-level data from the US- EPA's Toxic Release Inventory Program (TRI) and join it by spatial location to polygon-level data of census tract boundaries from census.gov.

QGIS and DBeaver will be used to manage the data. LucidCharts will help us easily sketch an Entity-Relationship Diagram. PostgresSQL with the PostGIS extension will be used to write code to process and analyze these data.

The TRI Program contains information about chemical releases from industrial and federal facilities. The data contains information about toxic releases with the following attributes:

- Facility name, address, latitude and longitude coordinates, and industry sector codes
- Chemical identification and classification information
- Quantities of chemicals released on site at the facility
- Quantities of chemicals transferred to Publicly Owned Treatment Works (POTWs)
- Quantities of chemicals transferred off site to other locations for release/disposal and further waste management
- Quantities of chemicals managed through disposal, energy recovery, recycling and treatment; non-production-related waste quantities; production/activity ratio

You can further review documentation about the TRI attributes here:

> https://www.epa.gov/toxics-release-inventory-tri-program/tri-basic-data-files-guide

Let's get to it!

In this tutorial, we'll use data for years 1987 to 2018. Start by downloading annual .csv data files for Pennsylvania TRIs here:

> https://www.epa.gov/toxics-release-inventory-tri-program/tri-basic-data-files-calendar-years-19872018

Once downloaded, open Command Prompt and go to the folder where all the .csv files were downloaded. Make sure all the .csv files are downloaded into the same folder. Once you're in the correct folder path, use the following code to combine all the .csv files into one named **TRI_PA_1987_2018_all_zips.csv**

```
copy *.csv TRI_PA_1987_2018_all_zips.csv
```

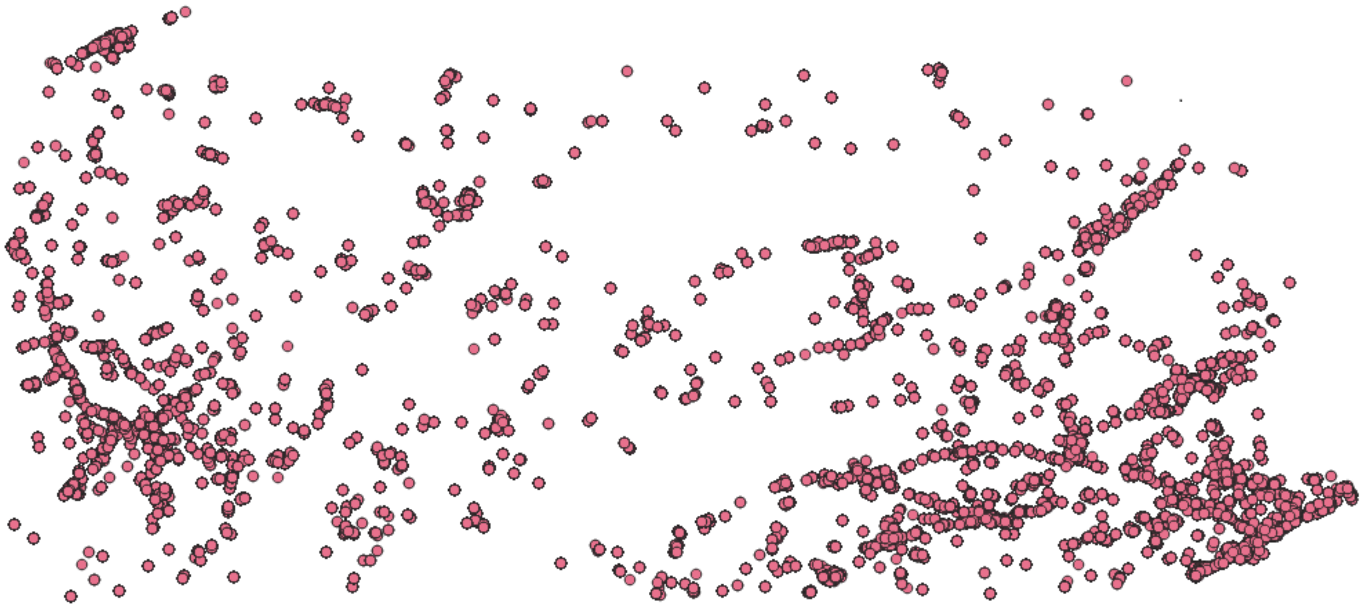
Next you will need to download QGIS, a freely available geographic information system, downloadable here:

> https://qgis.org/en/site/forusers/download.html

QGIS will allow us to convert the combined .csv file into a data format with queryable spatial geometries. This can be done by invoking the following commands:

Add Vector Layer > **Add Delimited Text Layer** >> import .csv file and designate it as **Point Coordinates** under the **Geometry Definition** section.

Once completed, you should now see a visualization of all 143,436 points in this Pennsylvania TRI dataset:

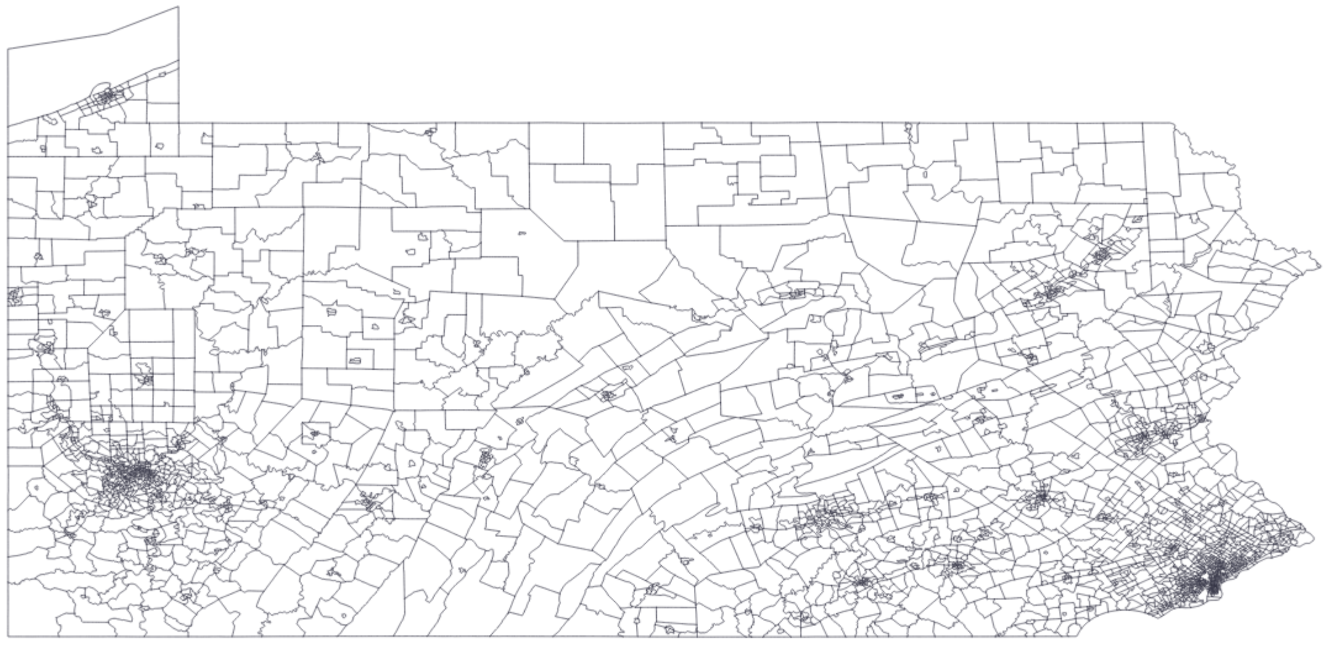


Repeat the above QGIS step with Pennsylvania census tract data. You have the option of importing a Shapefile instead of a .csv file using the following link:

> https://catalog.data.gov/dataset/tiger-line-shapefile-2016-state-pennsylvania-current-census-tract-state-based.

A shapefile is a common spatial data format which you may encounter when working with other datasets. If you download the shapefile, you will have a zipped folder containing several files. Keep this folder zipped. In QGIS go to **Add Vector Layer** > browse for the zipped folder, select it, and import it.

Once completed, you should see the 3,218 census tracts which make up Pennsylvania.



Next setup DBeaver, a SQL client and database administration tool which we will use to manage the TRI data and create entity-relationship tables. DBeaver is freely available through the following link: dbeaver.io.

Once DBeaver is setup, create a new database connection to PostgresSQL. PostgreSQL is a powerful open source object-relational database system which uses SQL as its coding language. The first thing you'll want to do is add the PostGIS extension. PostGIS is an open source software program that adds support for geographic objects to the PostgreSQL object-relational database.

To do this, type the following code into DBeaver's text editor:

```
create extension postgis;
```

You can read more about PostGIS functions here:

> postgis.net/workshops/postgis-intro/

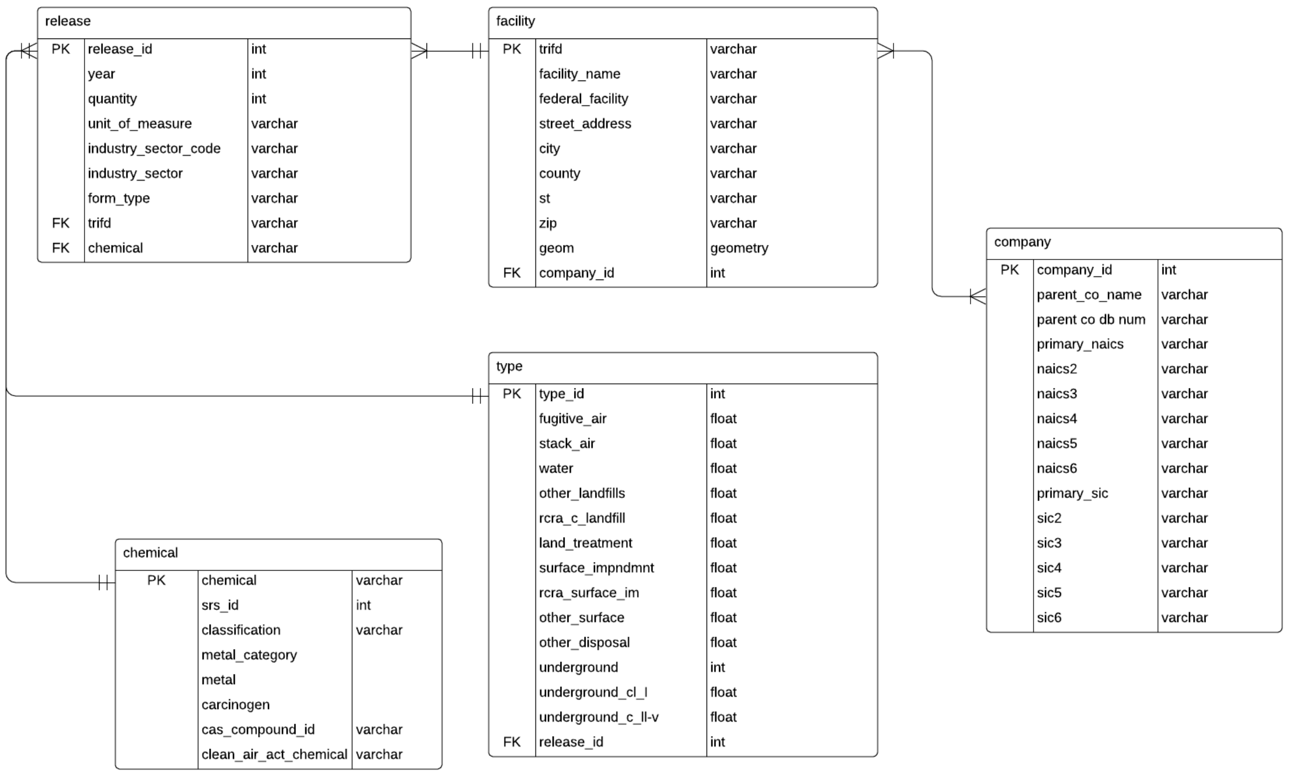
Great! Next we'll want to connect our newly created spatial data, stored in QGIS to DBeaver.

In order to do this, a new schema needs to first be created in DBeaver. Do this by right-clicking in the database navigator and choosing **create** > **schema** then naming it something relevant. Once created, go back to QGIS and connect to the schema by connecting to DB Manager under the database tab. You should now have your data in DBeaver.

Next, we will have to normalize our data. The data comes with over 110 fields, many of which can be calculated through spatial querying. For this tutorial, the TRI data was normalized into five entity-relationship tables, each containing a primary key for unique entries, as well as foriegn keys to connect to other tables' primary keys. lucidchart.com can be used to diagram these tables. The list below describes these tables in further detail.

1. **Facility**: Contains facility location information, geometries, address, name, and TRI emission identifier.
2. **Company**: Contains parent company information, NAICS and SIC information.
3. **Release**: Contains emission release quantity, unit of measure, year, industry code, and form type.
4. **Chemical**: Contains information about the chemical released such as its classifcation, carcinogen status, metal status, clean air act chemical status, and cas compound identifier.
5. **Type**: Contains information about the type of release such as air, water, underground, and surface.

The following image shows and Entity-Relationship-Diagram (ERD) of how the tables are connected. The entity connections use a crows feet convention, which can be further reviewed in lucidchart.com.



release

PK	release_id	int
	year	int
	quantity	int
	unit_of_measure	varchar
	industry_sector_code	varchar
	industry_sector	varchar
	form_type	varchar
FK	trifd	varchar
FK	chemical	varchar

facility

PK	trifd	varchar
	facility_name	varchar
	federal_facility	varchar
	street_address	varchar
	city	varchar
	county	varchar
	st	varchar
	zip	varchar
	geom	geometry
FK	company_id	int

company

PK	company_id	int
	parent_co_name	varchar
	parent co db num	varchar
	primary_naics	varchar
	naics2	varchar
	naics3	varchar
	naics4	varchar
	naics5	varchar
	naics6	varchar
	primary_sic	varchar
	sic2	varchar
	sic3	varchar
	sic4	varchar
	sic5	varchar
	sic6	varchar

type

PK	type_id	int
	fugitive_air	float
	stack_air	float
	water	float
	other_landfills	float
	rcra_c_landfill	float
	land_treatment	float
	surface_impndmnt	float
	rcra_surface_im	float
	other_surface	float
	other_disposal	float
	underground	int
	underground_cl_I	float
	underground_c_II-v	float
FK	release_id	int

chemical

PK	chemical	varchar
	srs_id	int
	classification	varchar
	metal_category	
	metal	
	carcinogen	
	cas_compound_id	varchar
	clean_air_act_chemical	varchar

Creating Tables in DBeaver

The following SQL script inputs the tables described above into DBeaver. Once created, we are ready to begin asking our spatial queries!

```
-- set the path and rename the tri table
set search_path to tri_pa_1987_2018, public;
alter table tri_pa_1987_2018."TRI_PA_1987_2018_all_zips"
rename to tri_pa_1987_2018;

select count(*)
from tri_pa_1987_2018;

--create the company table
drop table if exists company cascade;
create table company (
  company_id serial primary key,
  parent_co_name varchar,
  parent_co_db_num varchar,
  primary_naics varchar,
  naics2 varchar,
  naics3 varchar,
  naics4 varchar,
  naics5 varchar,
  naics6 varchar,
  primary_sic varchar,
  sic2 varchar,
  sic3 varchar,
  sic4 varchar,
  sic5 varchar,
  sic6 varchar
  );

insert into company (parent_co_name,
parent_co_db_num, primary_naics, naics2, naics3, naics4,
naics5, naics6, primary_sic, sic2, sic3, sic4, sic5, sic6)
select
distinct "112. parent co name" :: varchar,
"113. parent co db num" :: varchar,
"23. primary naics" :: varchar,
"24. naics 2" :: varchar,
"25. naics 3" :: varchar,
"26. naics 4" :: varchar,
"27. naics 5" :: varchar,
"28. naics 6" :: varchar,
"17. primary sic" :: varchar,
"18. sic 2" :: varchar,
"19. sic 3" :: varchar,
"20. sic 4" :: varchar,
"21. sic 5" :: varchar,
"22. sic 6" :: varchar
from tri_pa_1987_2018;

--create the facility table
drop table if exists facility cascade;
create table facility (
  trifd varchar primary key,
  facility_name varchar,
  federal_facility varchar,
  street_address varchar,
  city varchar,
  county varchar,
  st varchar,
  zip varchar,
  geom geometry,
  company_id int references company (company_id)
  );

insert into facility (trifd, facility_name,
federal_facility, street_address, city,
county, st, zip, geom, company_id)
select
distinct "2. trifd" :: varchar,
"4. facility name" :: varchar,
"14. federal facility" :: varchar,
"5. street address" :: varchar,
"6. city" :: varchar,
"7. county" :: varchar,
"8. st" :: varchar,
"9. zip" :: varchar,
geom :: geometry,
( select company_id
  from company as a
   where a.parent_co_db_num = b."113. parent co db num"
   limit 1)
from tri_pa_1987_2018 as b;
```

```
--create the chemical table
drop table if exists chemical cascade;
create table chemical (
  chemical varchar primary key,
  srs_id int,
  classification varchar,
  metal_category int,
  metal varchar,
  carcinogen varchar,
  cas_compound_id varchar,
  clean_air_act_chemical varchar
  );

insert into chemical (chemical, srs_id,
classification, metal_category, metal,
carcinogen, cas_compound_id, clean_air_act_chemical )
select
distinct "30. chemical" :: varchar,
"32. srs id" :: int,
"34. classification" :: varchar,
"36. metal category" :: int,
"35. metal" :: varchar,
"37. carcinogen" :: varchar,
"31. cas #/compound id" :: varchar,
"33. clean air act chemical" :: varchar
from tri_pa_1987_2018;


--create the release table
drop table if exists release cascade;
create table release (
  release_id serial primary key,
  year int,
  quantity int,
  unit_of_measure varchar,
  industry_sector_code varchar,
  industry_sector varchar,
  form_type varchar,
  trifd varchar references facility (trifd),
  chemical varchar references chemical (chemical)
  );
```

```
insert into release (year, quantity,
unit_of_measure, industry_sector_code,
industry_sector, form_type, trifd, chemical)
select
distinct  "1. year" :: int,
"54. on-site release total" :: int,
"39. unit of measure" :: varchar,
"15. industry sector code" :: varchar,
"16. industry sector" :: varchar,
"38. form type" :: varchar,
"2. trifd" :: varchar,
"30. chemical" :: varchar
from tri_pa_1987_2018;

--create the type table
drop table if exists type cascade;
create table type (
  type_id serial primary key,
  fugitive_air float,
  stack_air float,
  water float,
  other_landfills float,
  rcra_c_landfill float,
  land_treatment float,
  surface_impendmnt float,
  rcra_surface_im float,
  other_surface float,
  other_disposal float,
  underground int,
  underground_cl_l float,
  underground_c_ll_v float,
  release_id serial references release (release_id)
  );

insert into type (fugitive_air, stack_air, water,
other_landfills, rcra_c_landfill, land_treatment,
surface_impendmnt, rcra_surface_im, other_surface,
other_disposal, underground, underground_cl_l,
underground_c_ll_v, release_id )
select
distinct "40. 5.1 - fugitive air" :: float,
"41. 5.2 - stack air" :: float,
"42. 5.3 - water" :: float,
"48. 5.5.1b - other landfills" :: float,
"47. 5.5.1a - rcra c landfill" :: float,
"49. 5.5.2 - land treatment" :: float,
"50. 5.5.3 - surface impndmnt" :: float,
"51. 5.5.3a - rcra surface im" :: float,
"52. 5.5.3b - other surface i" :: float,
"53. 5.5.4 - other disposal" :: float,
"43. 5.4 - underground" :: float,
"44. 5.4.1 - underground cl i" :: float,
"45. 5.4.2 - underground c ii-v" :: float,
(select release_id
 from release as a
  where a.trifd = b."2. trifd"
  limit 1)
from tri_pa_1987_2018 as b;
```

Spatial Queries

You are now ready to ask an endless amount of spatial queries. Work through the following four examples. Once completed, read through the PostGIS documentation and try out new functions to play around with your own queries!

Spatial Query 1:

For the year 1990, what chemicals in which census tracts, along with their TRIFD identifier, reported their max emission as greater than 1,000,000?

Use the facility and release tables to join to the Census Tract table. Think about how to select chemical names, TRI emission identifiers (trifd), corresponding PA census tract, year, and emissions.

You will want to use an aggegrate function to choose the max emissions. Fortunately, we designed these tables to be relational. Join the facility table, which contains information about the facility's location to the release table which contains information about the emission. This join can be done by the trifd field which is the primary key of the facility table, and the foreign key of the release table.

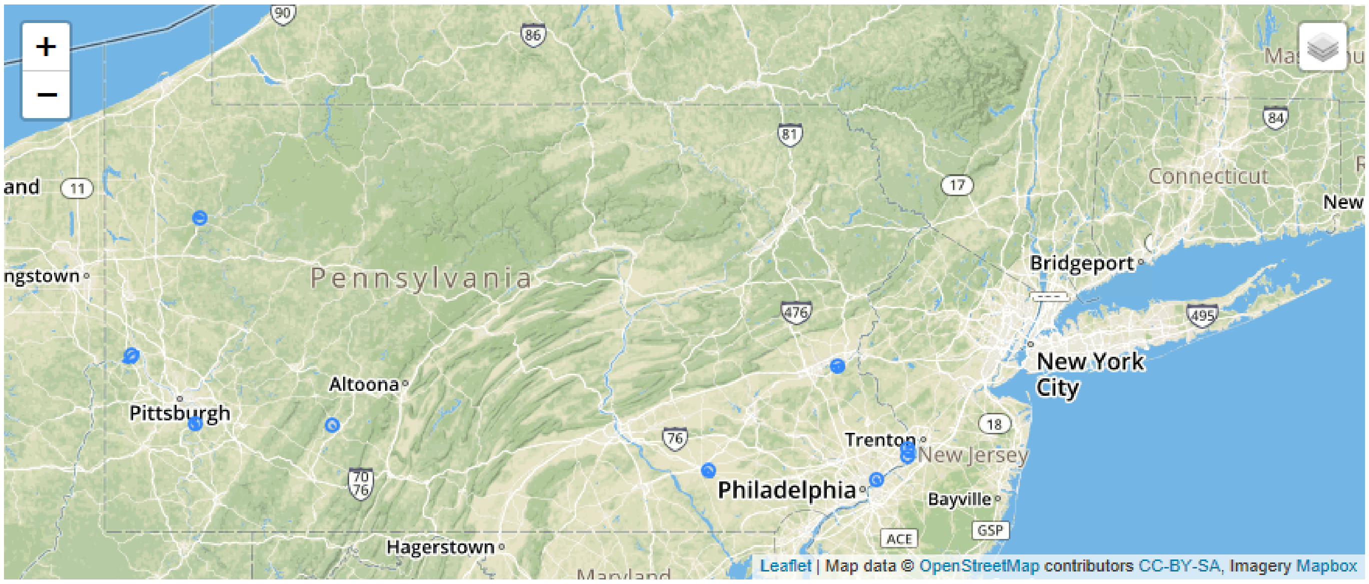
Next join to the census tract table by using a PostGIS function: *ST_Intersects* . This allows for a join by spatial location of the TRI points which intersect to each census tract polygon. Oftentimes, you may recieve spatial data with differing projection systems. *The ST_SRID(geom)* function allows us to check the geographic projection of each layer.

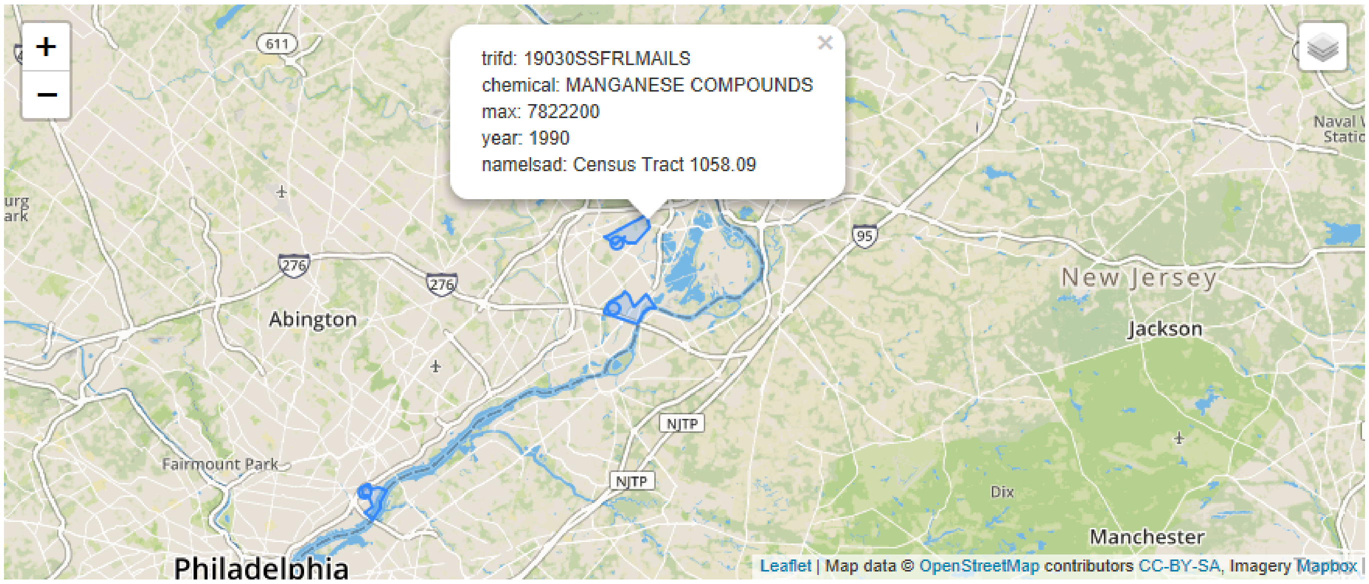
In our case, the The PA Census Tracts layer uses the SRID 4269 https://epsg.io/4269 coordinate system. The TRI data uses SRID 4326 https://epsg.io/4326 coordinate system . Since these projection systems differ, the geometry of the census tract layer has to be transformed to match the TRI facilities in order for this query to work, using the *ST_Transform* function.

```
select a.trifd,
  b.chemical,
  max(b.quantity) as max_emission_in_Census_Tract,
  b.year, c.namelsad,
  c.geom as Census_Tract_Geometries,
  a.geom as TRI_Geometries
from facility as a
join release as b
  on a.trifd = b.trifd
join
 tl_2016_42_tract as c
 on ST_Intersects(ST_Transform(c.geom, 4326)
 :: geography, a.geom)
where b.year = 1990
  and b.quantity > 1000000
group by a.trifd,
  b.chemical,
  c.namelsad,
  b.chemical,
  b.year,
  c.geom,
  a.geom;
```

RESULT:

```
trifd          |chemical             |max_emission_in_census_tract|year|namelsad
---------------|---------------------|----------------------------|----|-----------
15025SSCLR400ST|AMMONIA              |                     1140000|1990|Census Trac
15061ZNCCR300FR|ZINC COMPOUNDS       |                     1310750|1990|Census Trac
15902JHNST545CE|CHROMIUM             |                     1141510|1990|Census Trac
16323FRNKL600AT|1,1,1-TRICHLOROETHANE|                     1243000|1990|Census Trac
17601RRDNN216GR|TOLUENE              |                     1182000|1990|Census Trac
18016BTHLH501EA|ZINC COMPOUNDS       |                     1070600|1990|Census Trac
19007MCMPNGREEN|TOLUENE              |                     1900250|1990|Census Trac
19030SSFRLMAILS|MANGANESE COMPOUNDS  |                     7822200|1990|Census Trac
19137LLDSGMARGA|CUMENE               |                     1053700|1990|Census Trac
```




trifd: 19030SSFRLMAILS
chemical: MANGANESE COMPOUNDS
max: 7822200
year: 1990
namelsad: Census Tract 1058.09
Abington
Philadelphia
Fairmount Park
New Jersey
Jackson
Manchester
Dix
Leaflet | Map data © OpenStreetMap contributors CC-BY-SA, Imagery Mapbox

Spatial Query 2:

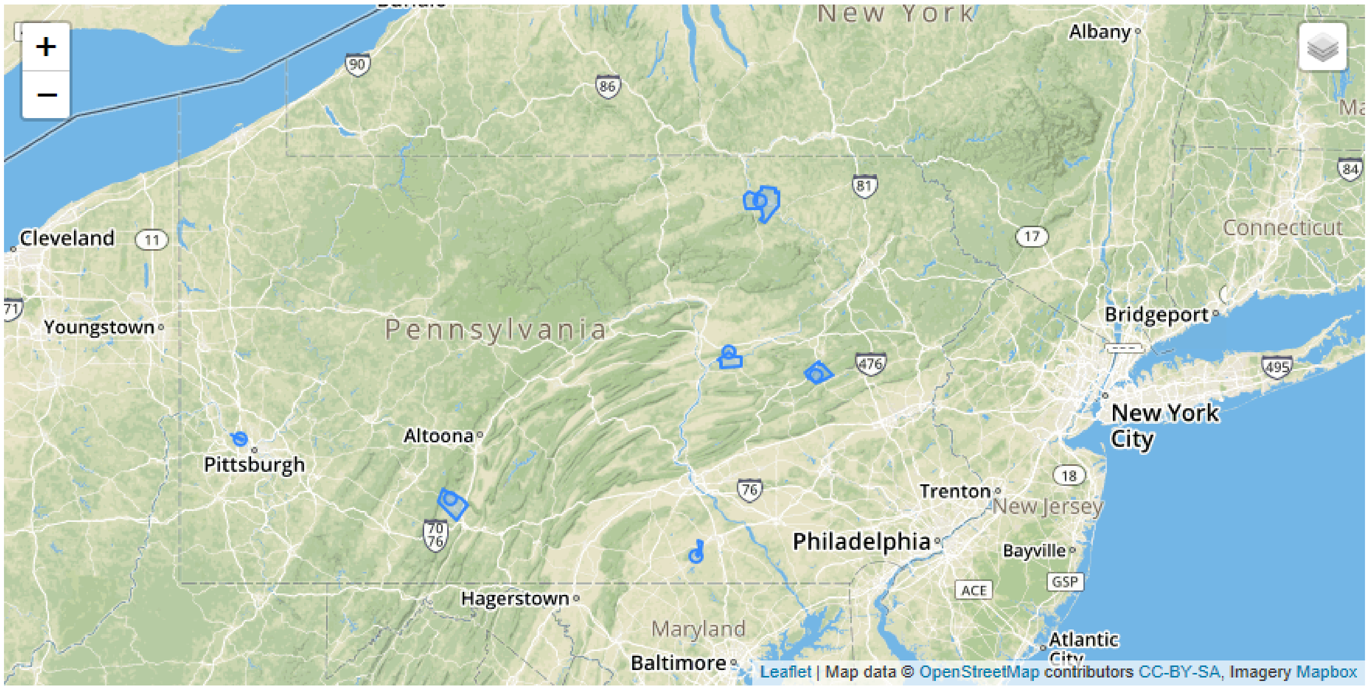
List the census tracts which contain chemicals that are classified as a carcinogen and that emitted between 10,000 and 15,000 pounds for the year 2018.

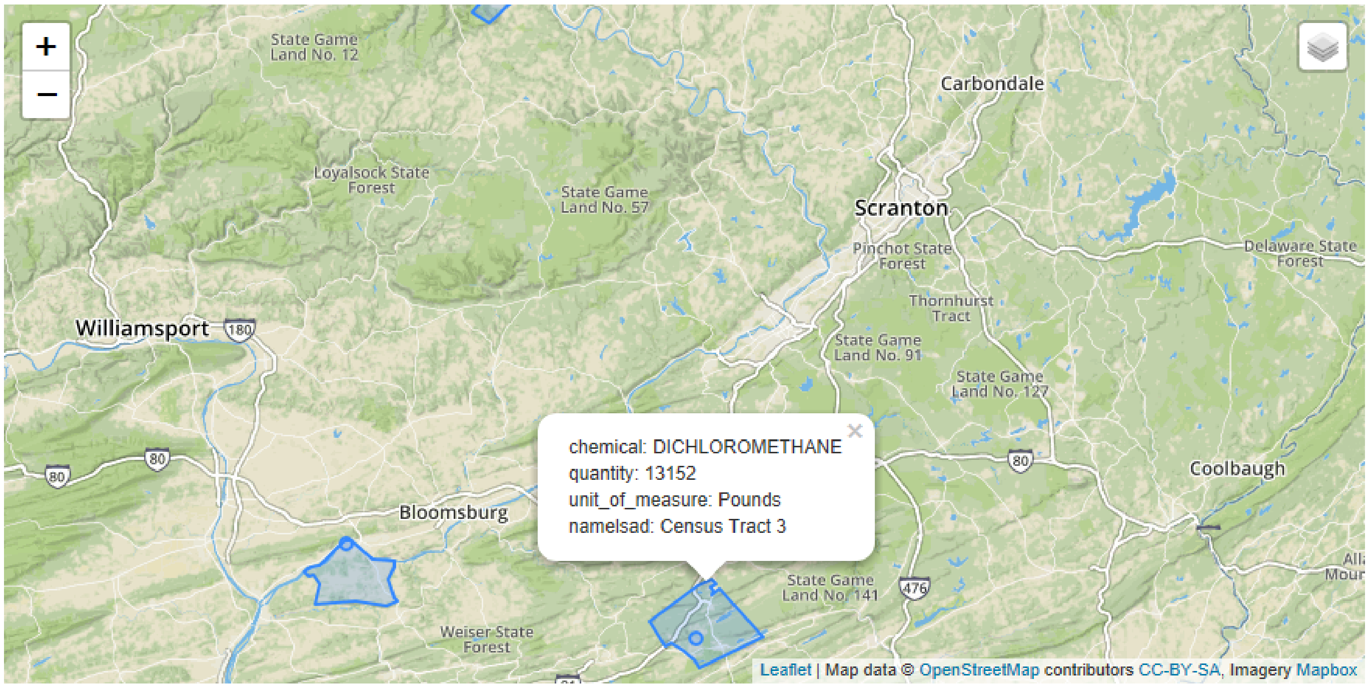
The second spatial query uses the facility, chemical, release, and PA Census Tracts tables. In this query, we're interested in the names of the chemicals, their emission quantity between 10,000 and 15,000, emission unit of measure, and census tract name, for emissions which are classified as a carcinogen in the year 2018. This query requires identifying all the TRI points which intersect each census tract layer. Again, since these projection systems were different, the geometry of the census tract layer had to be transformed to match the TRI facilities in order for this query to work.

```
select a.chemical,
  a.quantity,
  a.unit_of_measure,
  d.namelsad,
  d.geom as Census_Tract_Geometries,
  c.geom as TRI_Geometries
from release as a
join chemical as b
        on a.chemical = b.chemical
join facility as c
        on a.trifd = c.trifd
join
   tl_2016_42_tract as d
   on ST_Intersects(ST_Transform(d.geom, 4326)
   :: geography, c.geom)
where b.carcinogen ilike 'YES'
  and a.year = 2018
  and a.quantity between 10000 and 15000
group by a.chemical,
  a.quantity,
  a.unit_of_measure,
  d.namelsad,
  d.geom,
  c.geom;
```

RESULT:

```
chemical       |quantity|unit_of_measure|namelsad           |census_tract_geometrie
---------------|--------|---------------|-------------------|----------------------
ACETALDEHYDE   |   12909|Pounds         |Census Tract 205.21|MULTIPOLYGON (((-76.87
ACETALDEHYDE   |   13314|Pounds         |Census Tract 9509  |MULTIPOLYGON (((-76.52
DICHLOROMETHANE|   13152|Pounds         |Census Tract 3     |MULTIPOLYGON (((-76.16
DICHLOROMETHANE|   13407|Pounds         |Census Tract 807   |MULTIPOLYGON (((-76.71
STYRENE        |   11517|Pounds         |Census Tract 4610  |MULTIPOLYGON (((-80.16
STYRENE        |   13176|Pounds         |Census Tract 9603  |MULTIPOLYGON (((-78.69
```





chemical: DICHLOROMETHANE
quantity: 13152
unit_of_measure: Pounds
namelsad: Census Tract 3

Spatial Query 3:

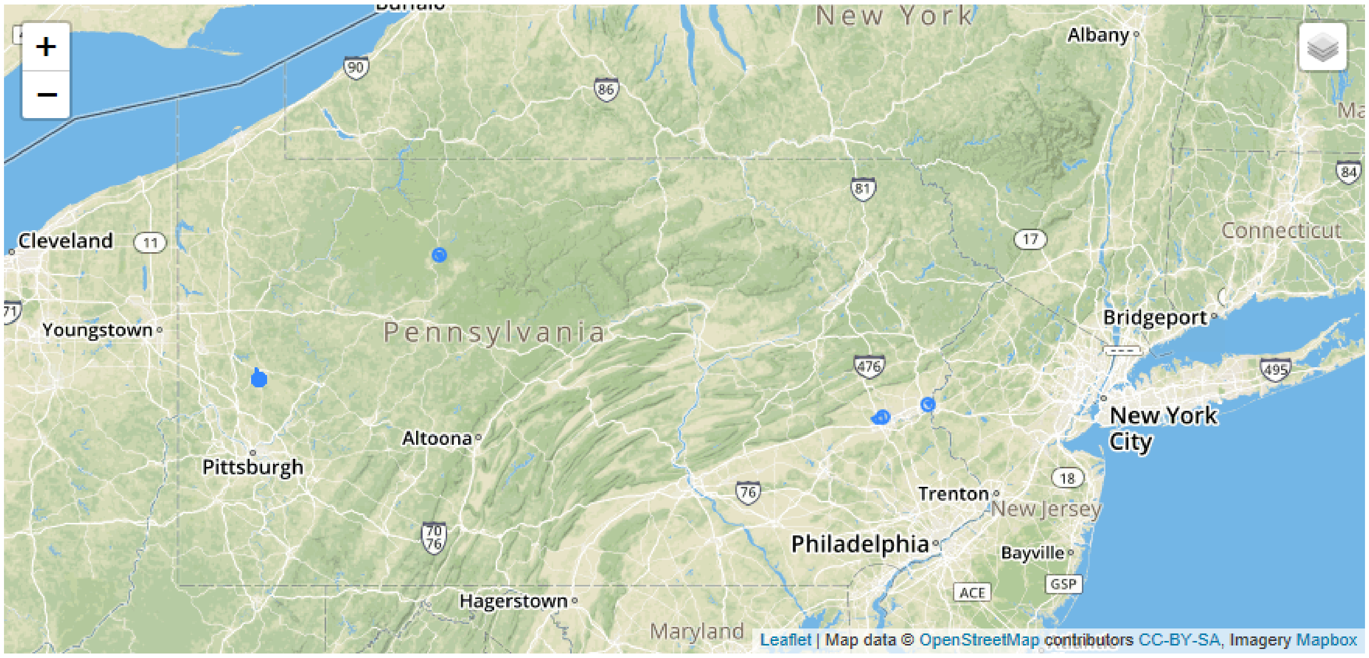
For all years, identify the chemicals along with their corresponding census tracts that released greater than 4,500,000 pounds into water. Provide your answer in descending order.

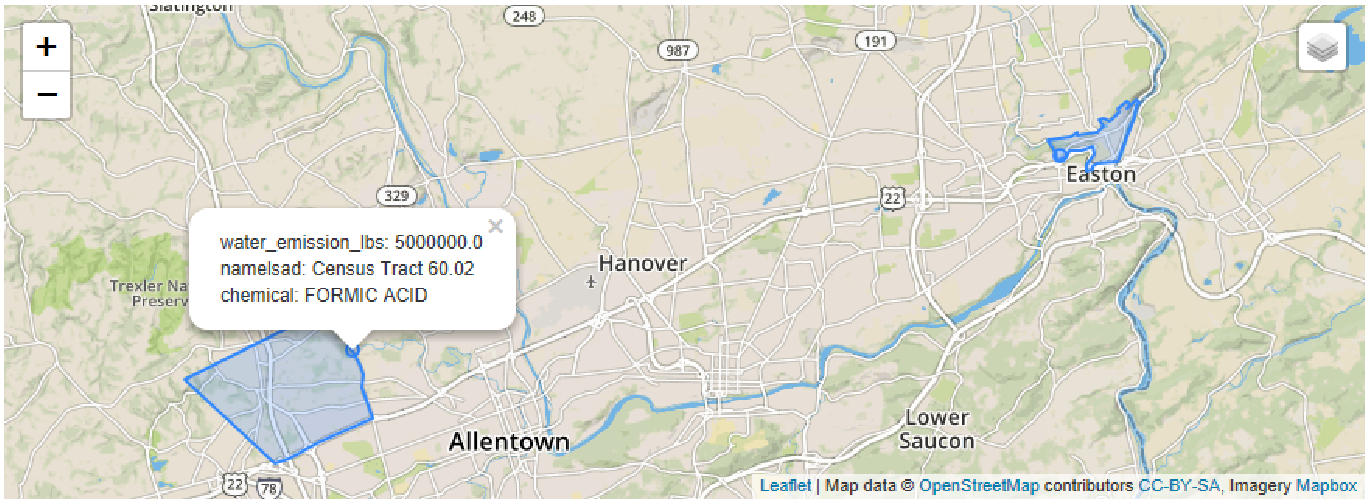
The third spatial query uses four tables: type, release, facility, and PA Census Tracts. In this query, we're interested in the names of the chemicals which are emitted into water at over 4,500,000 pounds. Remember to order the output in descending order and transform the geography to 4326.

```
select a.water as water_emission_lbs,
  d.namelsad,
  b.chemical as chemical_name,
  c.geom as TRI_Geometries,
  d.geom as Census_Tract_Geometries
from type as a
join release as b
  on a.release_id = b.release_id
join facility as c
  on b.trifd = c.trifd
join
   tl_2016_42_tract as d
   on ST_Intersects(ST_Transform(d.geom, 4326)
   :: geography, c.geom)
where a.water > 4500000
order by d.namelsad, b.chemical, a.water desc;
```

RESULT:

```
water_emission_lbs|namelsad          |chemical_name
------------------|------------------|---------------------------------------------
         6003084.0|Census Tract 141  |SULFURIC ACID (1994 AND AFTER ACID AEROSOLS"
         5000000.0|Census Tract 60.02|FORMIC ACID
             3.2E7|Census Tract 9030 |ZINC COMPOUNDS
             3.1E7|Census Tract 9030 |ZINC COMPOUNDS
             2.8E7|Census Tract 9030 |ZINC COMPOUNDS
             2.6E7|Census Tract 9030 |ZINC COMPOUNDS
             1.2E7|Census Tract 9030 |ZINC COMPOUNDS
         9800000.0|Census Tract 9030 |ZINC COMPOUNDS
         8400000.0|Census Tract 9030 |ZINC COMPOUNDS
         7560000.0|Census Tract 9505 |CATECHOL
```




water_emission_lbs: 5000000.0
namelsad: Census Tract 60.02
chemical: FORMIC ACID
Hanover
Easton
Allentown
Lower Saucon
Leaflet | Map data © OpenStreetMap contributors CC-BY-SA, Imagery Mapbox

Spatial Query 4:

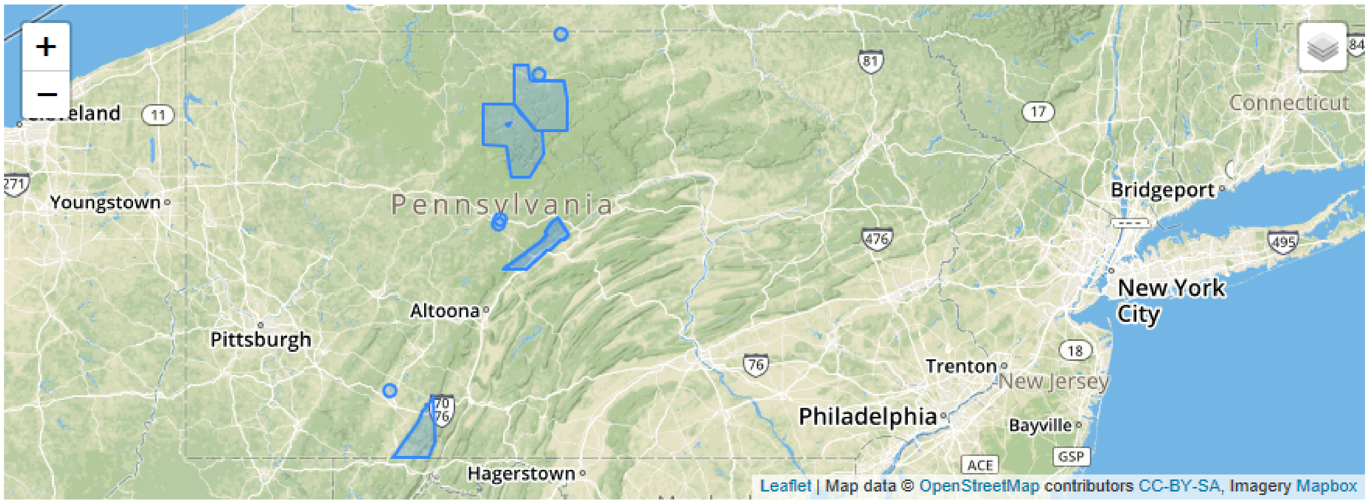
Perhaps you may be interested in knowing which TRI emissions are near a census tract but do not actually fall within their geographic boundaries. Find the top five closest TRI emissions to each census tract, but limit to the top 5 census tracts with the farthest nearest neighbor. Report in meters and in descending order.

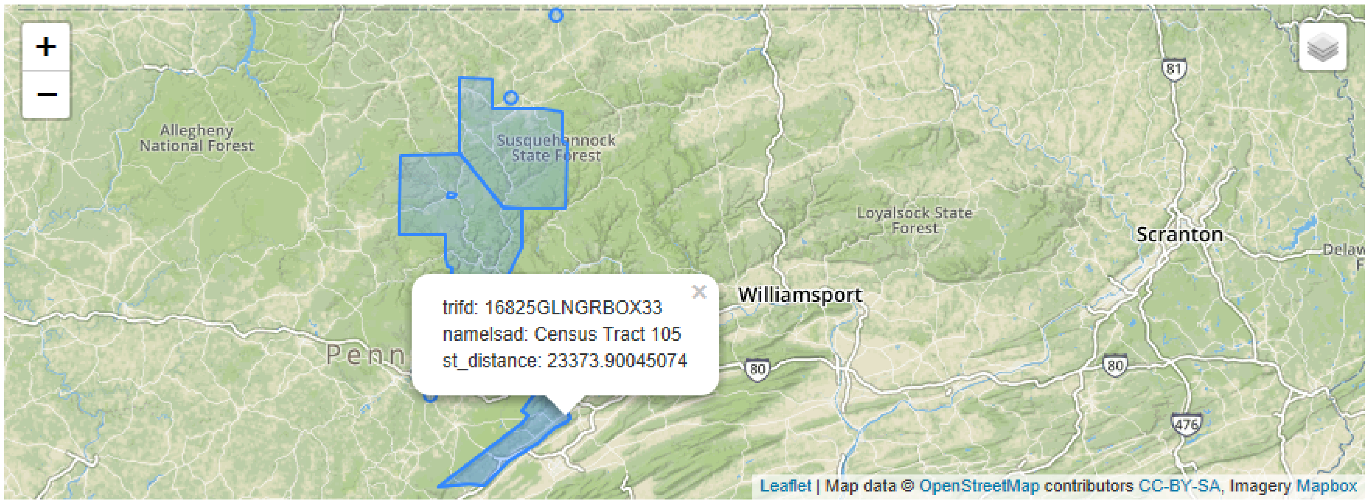
This fourth spatial query uses a KNN operator '<#>' to calculate the distance, of the nearest TRI emission to each census tract. This query uses a cross lateral join, which allows rows from the census tract table to assume multiple values from the TRI emissions table. In this join, we can set a limit to identify the closest 5 TRI emissions to each census tract, rather than generate a list of distances of ALL emissions to ALL census tracts, which would have been a time-intensive and massive query. This query was then further refined to limit to the 5 "farthest" nearest neighbors.

```
select a.trifd, c.namelsad,
  ST_Distance(ST_Transform(c.geom, 4326)
    :: geography, a.geom) as Distance_in_Meters,
  c.geom as Census_Tract_Geometries,
  a.geom as TRI_Geometries
from facility as a
cross join lateral
  (select b.namelsad, b.geom
    from tl_2016_42_tract as b
    order by b.geom <#> a.geom limit 5) as c
 order by Distance_in_Meters desc limit 5;
```

RESULT:

```
trifd          |namelsad         |distance_in_meters|census_tract_geometries
---------------|-----------------|------------------|------------------------------
16923HRBRTGOLDR|Census Tract 9504|    26363.68680118|MULTIPOLYGON (((-78.205353 41.
16825BRDFRMCDOW|Census Tract 105 |    24821.18701032|MULTIPOLYGON (((-78.282781 40.
15501PBSQC10310|Census Tract 213 |     23488.2263498|MULTIPOLYGON (((-79.055957999
16825GLNGRBOX33|Census Tract 105 |    23373.90045074|MULTIPOLYGON (((-78.282781 40.
16915LHLNC203CH|Census Tract 9602|    22350.41555333|MULTIPOLYGON (((-78.420968 41.
```




Allegheny National Forest
Susquehannock State Forest
Loyalsock State Forest
Scranton
Williamsport
Leaflet | Map data © OpenStreetMap contributors CC-BY-SA, Imagery Mapbox

trifd: 16825GLNGRBOX33
namelsad: Census Tract 105
st_distance: 23373.90045074

Improving Query Optimization

Fortunately, these datasets are relatively small in the cosmic universe of spatial information. However, you may encounter datasets with hundreds of millions of rows, or you may ask complicated queries which take a lot of computing power to read and write operations. The KNN operator for the fourth query could generate a massive output if no limits were set. To optimize you may want to construct an index or denormalize the data. If you find that the time it takes to execute a query takes a long time (could be minutes), then you may want to create an index or denormalize.

Index Construction

There are tools available online like SQL Profiler which can help you identify queries which are peforming at suboptimal levels:

> https://www.sqlmvp.org/sql-server-profiler-trace/

Either way, an index allows a query to search data quickly by searching part of a table rather than the entire dataset. Try constructing an index using the following format:

```
create index index_name
on table_name(column_to_index);
```

Denormalization

The ERD shown earlier shows how the TRI data have been normalized into five tables. The purpose of this is to reduce redundancy and eliminate deletion and insertion errors. However you may want to minimize invoking several joins to get to a certain attribute. You may have noticed that the geom is only found in the facility table. You may not want to always include the facility table in your queries. However at the moment, you cannot avoid joining to it if you want to access the geom. Go ahead and try adding the GEOM field to the other four TRI tables, then get on out there and create some new queries!

I hope you enjoyed this tutorial!

Tom McKeon, MPH