PostGIS SQL Cheatsheet
GIS with SQL

Using PostGIS, it's possible to do much of what you would usually do in a GIS in the database using SQL. This can be helpful because SQL is repeatable—it's text that you can copy and re-run later as needed. In an interactive context, you might take user input such as a map point or something more complex like a polygon a user draws and use these geometries in an SQL query.

When you use PostGIS, you access GIS functionality usually through special functions that begin with `ST_`. You can find the PostGIS documentation, which includes each of these special functions, online at postgis.net/docs/.

Carto uses PostGIS, so any PostGIS functions will be available through Carto. PostGIS is open source and can be installed on your computer or on a server for use in a web application.

Common Queries

Select polygons' areas

```
SELECT *, ST_area(the_geom::geography) AS area
FROM table
```

For example:

```
SELECT *, ST_area(the_geom::geography) AS area
FROM countries
```

This query selects all of the columns from the table (`*`) and appends a new column (`area`) that contains the area of the features in square meters. Adding `::geography` to a geometry column asks PostGIS to consider the geometry in terms of the globe rather than projected coordinates. This ensures that our units are meters.

Select polygons' areas with a specific projection

```
SELECT *, ST_area(ST_transform(the_geom, 2263)) AS area
FROM table
```

For example:

```
SELECT *, ST_area(ST_transform(the_geom, 2263)) AS area
FROM nyc_census_tracts
```

This query is very similar to the previous query, but we are using `ST_transform` to reproject the geometry into EPSG:2263, first, which ensures that the units will be the units of the projection, in this case square feet.

Select features within a distance of a specified point

```
SELECT *
FROM table
WHERE ST_DWithin(
  the_geom::geography,
  cdb_latlng(latitude, longitude)::geography,
  distance
)
```

For example:

```
SELECT *
FROM dams
WHERE ST_DWithin(
  the_geom::geography,
  cdb_latlng(40.735, -73.994)::geography,
  100000
)
```

This query creates a point using `cdb_latlng()` (this is specific to Carto), then uses `ST_DWithin()` to find features within some distance of that point, in meters. In this example the database will return `dam` features within 100km of (`40.735, -73.994`).

Select features in a bounding box

```
SELECT *
FROM table
WHERE ST_within(
  the_geom_webmercator,
  ST_transform(
    ST_MakeEnvelope(
      min_lng, min_lat, max_lng, max_lat,
      4326
    ),
    3857
  )
)
```

For example:

```
SELECT *
FROM dams_copy
WHERE ST_within(
  the_geom_webmercator,
  ST_transform(
    ST_MakeEnvelope(
      -75, 40, -70, 45,
      4326
    ),
     3857
  )
)
```

This query selects features within a given bounding box, which would be specified in the italicized parts (minimum longitude, minimum latitude, maximum longitude, maximum latitude). The new functions here are `ST_within`, which checks if one geometry is *within* another, and `ST_MakeEnvelope`, which makes a rectangle from the given minimums and maximums.

Select buffered geometries

```
SELECT cartodb_id,
  ST_transform(
    ST_buffer(the_geom_webmercator, buffer_radius),
    3857
  ) as the_geom_webmercator
FROM table
```

For example:

```
SELECT cartodb_id,
  ST_transform(
    ST_buffer(the_geom_webmercator, 20000),
    3857
  ) as the_geom_webmercator
FROM dams
```

This query buffers your feature's geometries by the specified buffer radius (in the example, 20km) using `ST_buffer()`. Note that after we buffer the geometry we reproject the result into EPSG:3857 (webmercator, the projection webmaps use) and give it the name `the_geom_webmercator`. If we didn't do this, there would be no column for Carto to map. Note also that once we start specifying columns we would need to specify any other columns we want to have available in our popups or styles—here we only include `cartodb_id`, but you would include others as necessary.

Order features by their distance to a point

```
SELECT *
FROM table
ORDER BY the_geom <-> cdb_latlng(latitude, longitude)
```

```
LIMIT count
```

For example:

```
SELECT *
FROM dams
ORDER BY the_geom <-> cdb_latlng(40.735, -73.994)
LIMIT 10
```

This query creates a point using `cdb_latlng()` (this is specific to Carto), then uses `<->` to order the features by their distance from that point. In this example the database will return the 10 closest `dam` features to (`40.735, -73.994`).

Spatial join: select features that overlap with features in another table

```
SELECT t1.*
FROM table1 t1
LEFT OUTER JOIN table2 t2 ON
  ST_within(t1.the_geom, t2.the_geom)
WHERE condition
```

For example:

```
SELECT d.*
FROM dams d
LEFT OUTER JOIN countries c ON
  ST_within(d.the_geom, c.the_geom)
WHERE c.name = 'Canada'
```

This query selects features from one table that overlap with some features from another table using a spatial join. Some GISs refer to this as a “select by location.” It uses `ST_within` to check if a feature from `table1` is in a feature from `table2`, and if there is a match you can use the matching attributes in the `WHERE` clause. The example does this—it uses the `name` field from the `countries` table to pick which `dams` to select.

Spatial join: count features in polygons

```
SELECT t1.*, count(t2.*) AS t2_count
FROM table1 t1
LEFT OUTER JOIN table2 t2 ON
  ST_within(t2.the_geom, t1.the_geom)
GROUP BY t1.cartodb_id
```

For example:

```
SELECT c.*, count(d.*) AS dams_count
FROM countries c
```

```
LEFT OUTER JOIN dams d ON
  ST_within(d.the_geom, c.the_geom)
GROUP BY c.cartodb_id
```

This query selects all of the features from one table and counts overlapping features from another table using a spatial join. This is equivalent to counting points (or small polygons and lines) in polygons in a GIS. It uses `ST_within` to check if a feature from `table2` is in a feature from `table1`, and if there is a match it adds that feature to `t2_count`. The example does this and counts the number of `dams` within each feature in `countries`, and you could then use the `dams_count` field when styling the layer.