Spatial Database Design Final Report

Jin Xu

2018/12/15

Introduction

Objective

In this project, I am interested in mapping out the spatial distributions of most frequently updated buildings and roads using the Manhattan OSM history dump data.

Spatial Questions

- Which buildings or roads are the most frequently updated? What types of buildings or roads are they? Where do they locate? What is the range of them?
- Is there a difference in the range of updates for buildings or roads for each contributor? Is there a relation between contributors' total update and the range of updates?
- What kind of road is the nearest to the most frequently updated buildings? What kind of building the nearest to the most frequently updated roads?

Dataset and ERD

OSM history dump downloaded from: https://planet.openstreetmap.org/planet/full-history/ contains all information of global OSM features, including OSM feature id, updated times, users who made that update, and all tags related to this feature.
In this study, I downloaded the global OSM history dump data in a pbf format and clipped it by the boundary (N: 40.88 S: 40.68 E: -73.90 W: -74.04) to get the history dump data for only Manhattan. Later, I imported it into PostGIS database with the osm2pgsql tool. It automatically created tables shown in figure 1, and more specific process is included in Appendix 1.

The original pbf data is not a relational database, which means there is no tables or columns. The tables imported by osm2pgsql contains the feature id stored as bigint, tags stored as hstore, ways as geometry, and other columns. Buildings and roads are considered to be the mostly updated features in OSM, and they can be identified by the tags related to them. In this case, they will be features in the polygon and line table with values in the "building" or "highway" columns.

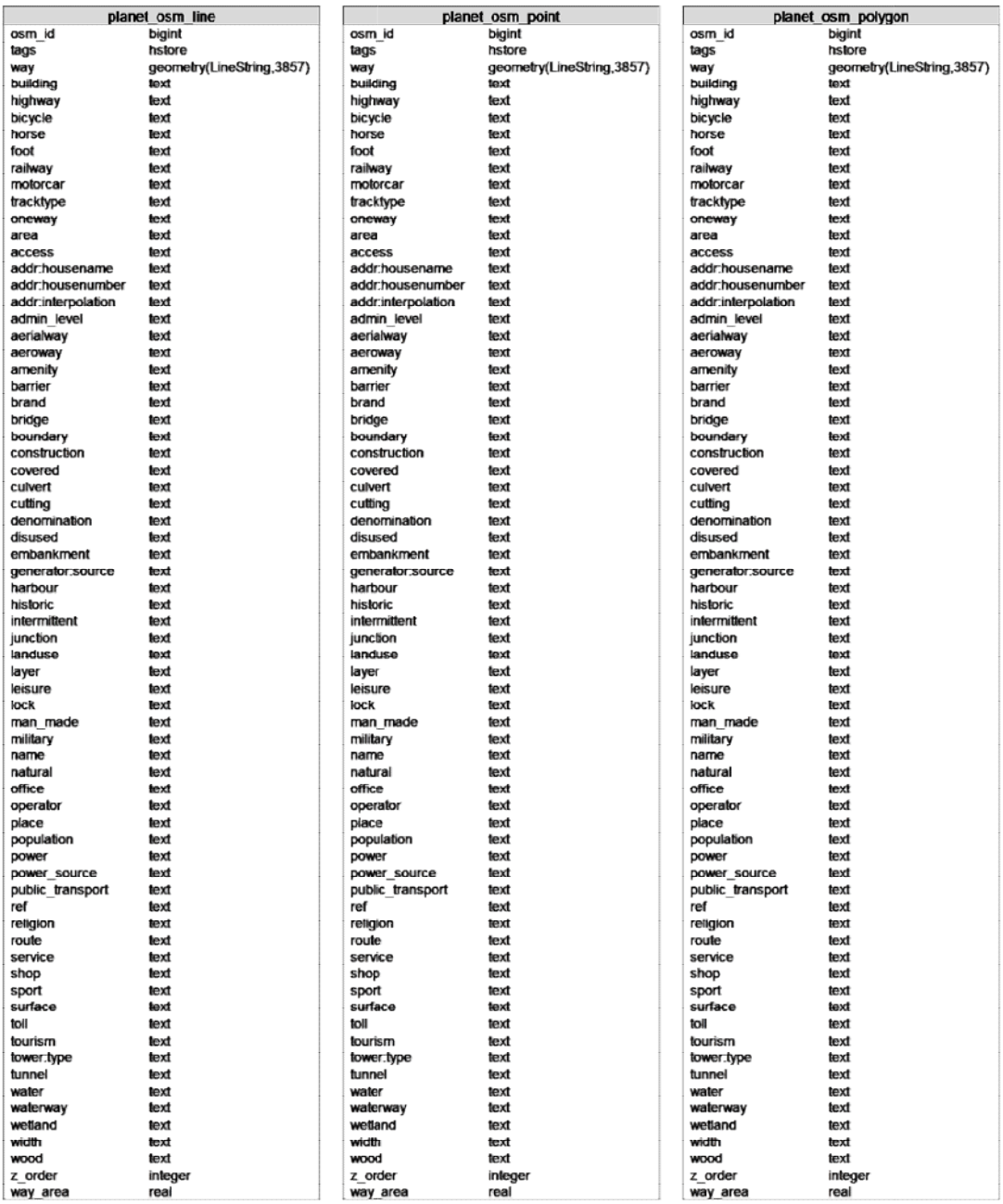


Figure 1. Current Database ERD

Database Structure and Normalization

ERD

The targeted ERD is shown in Figure 2. It mainly two tables, road and building table. Other tables serves as the look up table for reducing the data size of these two tables. The data dictionary is included in Appendix 4.

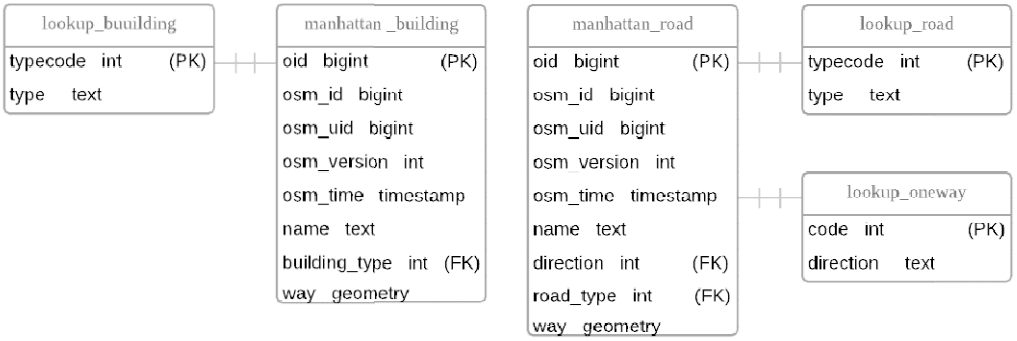


Figure 2. Target Database ERD

Normalization Process

The original dataset is not a relational database. Therefore I need to create one by myself. The type of building is casual so I need to re-code it based on the classification of buildings when OSM introduce it on： https://wiki.openstreetmap.org/wiki/Map_Features#Building. The process for normalization is shown as below:

1. create a look up table for clarifying the types.

2. select buildings from the polygon table in original database based on the building column is not null or “no”, as it is a rule for OSM data to distinguish other polygons from buildings.
3. for the purpose of this study, also select other columns that might contribute to the reclassification.

4. reclassification and re-code the type column.

5. repeat for road tables.

6. assign constrains and references.

The complete code is included in Appendix 2.

Optimization

Optimization code is included in Appendix 3

Create Index

The first method I applied for optimization is creating indexes. I created indexes on oid(PK), osm_uid, and geometry column for both tables as they are used as key columns when linking two tables.

After getting these indexes, process time to run the query of getting the updates and range for each contributors reduced by 1.1s. But the process time for getting the distance between different types of roads and the most frequently updated buildings increased by 1.8s.

Denormalization

The time to run the query of getting distance between different types of buildings and the most frequently updated roads are still more than 30 min. So I tried to denormalize the schema by saving the result of every steps into new tables and create indexes directly on related columns in these tables.

This strategy is as follow:

1. put data into new tables temporarily for join. In this case, it is the most frequently updated building and road type table.

2. create indexes directly on these two tables on geometry column for linking these two tables.

3. create another index for building type table for linking to the look up table.

4. calculate the average distance while joining these two tables without linking to the look up table. It might take less time in doing so because building type column is originally stored as numeric data, while the it would be replaced by characters if linked, which takes up much space for storing and processing.

5. save the final result into a table as it will reduce the time to show.

6. join to the look up table.

7. drop all tables after export.

The optimized query takes few seconds for other steps except for the join part. It took about 2 hours to finish the join. There have to be other ways for optimization.

Polygon to Point

It would be possible that the geometry of polygon takes up too much space for process. Would it be faster if I only calculate the distance between roads and the centroid of polygon?

st_distance(geometry) calculates the closest distance between buildings and roads but not the distance to the centroid. So the result for the calculated distance between lines and polygons will be shorter than than distance between lines and the centroid of polygons. But I am only calculated the distance for comparing between road types, the differences will apply for all types of roads and it would still be a fair comparison.

In ArcGIS, there is a function of turning polygons to points based on the centroid. This function in PostGIS is called st_centriod(geometry). It took few seconds to run every queries and the join query took 43 min to run when polygons were turned into points. Comparing to the 2 hour processing time, the speed for process is twice as much as the previous one.

The result before and after using st_centroid(geometry) is shown in figure 1. It seems that the result is increasing by different degree, but the ranking of data remains. Therefore, it would be a fair comparison if it only concerns with the ranking but not the number, and the value of the result is big enough to be resilience to this sort of error.

	building_type integer	avg double precision
1	0	10809.7485156912
2	1	9414.8551007685
3	2	9510.88996965078
4	3	10129.9315142547
5	4	8897.02499739807
6	99	10876.5611992156

Before

	building_type integer	avg double precision
1	0	10822.683809391
2	1	9437.23488957649
3	2	9543.9088188615
4	3	10160.6506313986
5	4	8921.48562483083
6	99	10886.4950107466

After

Figure 3. Result Before and After Optimization

Analyst Result

Most Frequently Updated Building and Road, Types, and Range

- Which buildings or roads are the most frequently updated? What types of buildings or roads are they? Where do they locate? What is the range of them?

Frequency is calculated as the change of version per day. The most frequently updated building or road is defined as the building or road with frequency of updates exceeding the average. Because the distribution of frequency of updates shows in a long-tail distribution, I used head/tail break for data classification. First, calculate the mean of all data, and classify data exceeding the mean as head and the rest as tail. Then, repeat for these steps for classification until the head no longer distribute as long-tail. But for the purpose of this study, I calculated the mean for only once.

Here is the query for selecting the most frequently updated buildings and their type.

```
SET search_path TO term_project, public;

--MOST FREQUENT BUILDING--
SELECT *
FROM(
     --calculate frequency
     SELECT osm_id, b.type,
            CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                 THEN (max(osm_version) - min (osm_version))
                      /(date_part('day', max(osm_time) - min(osm_time)))
            ELSE NULL
            END AS frequency,
            way
     FROM manhattan_building as a
     JOIN lookup_building as b
     ON a.building_type = b.typecode
     GROUP BY osm_id, b.type, way
     ORDER BY frequency DESC
) c
WHERE frequency > (
                    --calculate avg(frequency)
                    SELECT avg(frequency)
                    FROM(
                         SELECT osm_id, b.type,
                         CASE WHEN date_part('day', max(osm_time) - min(osm_time)) >
 0
                              THEN (max(osm_version) - min (osm_version))
                                   /(date_part('day', max(osm_time) - min(osm_tim
e)))
                         ELSE NULL
                         END AS frequency,
                         way
                         FROM manhattan_building as a
                         JOIN lookup_building as b
                         ON a.building_type = b.typecode
                         GROUP BY osm_id, b.type, way
                         ORDER BY frequency DESC
```

```
                        ) d
)
;
```

I constructed the query in these steps:

1. calculate the frequency of updates for each buildings based on their osm_id.

2. calculate the average frequency of update.

3. select buildings with updates exceeding the average, by comparing the frequency of updates in table selected in step 1 with the average frequency.

The result is latter mapped in figure 4.

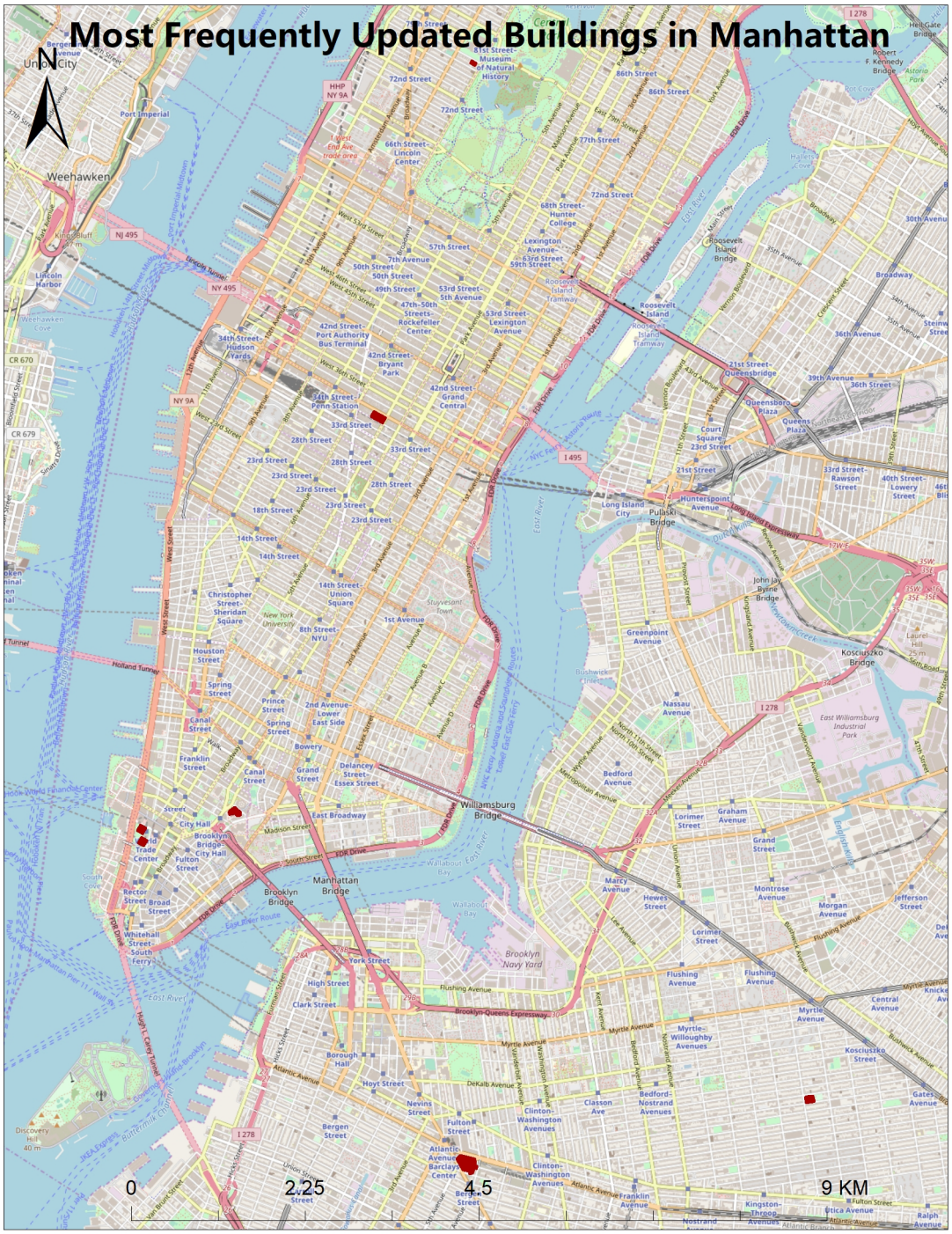


Figure 4. Most Frequently Updated Building in Manhattan

Similarly, I constructed the query for roads and the result is is mapped in figure 5.

```
--MOST FREQUENT ROADS--
SELECT *
FROM(
     --calculate frequency
     SELECT osm_id, b.type,
            CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                 THEN (max(osm_version) - min (osm_version))
```

```
                        /(date_part('day', max(osm_time) - min(osm_time)))
            ELSE NULL
            END AS frequency,
            way
     FROM manhattan_road as a
     JOIN lookup_road as b
     ON a.road_type = b.typecode
     GROUP BY osm_id, b.type, way
     ORDER BY frequency DESC
) c
WHERE frequency > (
                    --calculate avg(frequency)
                    SELECT avg(frequency)
                    FROM(
                         SELECT osm_id, b.type,
                         CASE WHEN date_part('day', max(osm_time) - min(osm_time)) >
 0
                              THEN (max(osm_version) - min (osm_version))
                                   /(date_part('day', max(osm_time) - min(osm_tim
e)))
                         ELSE NULL
                         END AS frequency,
                         way
                         FROM manhattan_road as a
                         JOIN lookup_road as b
                         ON a.road_type = b.typecode
                         GROUP BY osm_id, b.type, way
                         ORDER BY frequency DESC
                         ) d
)
;
```

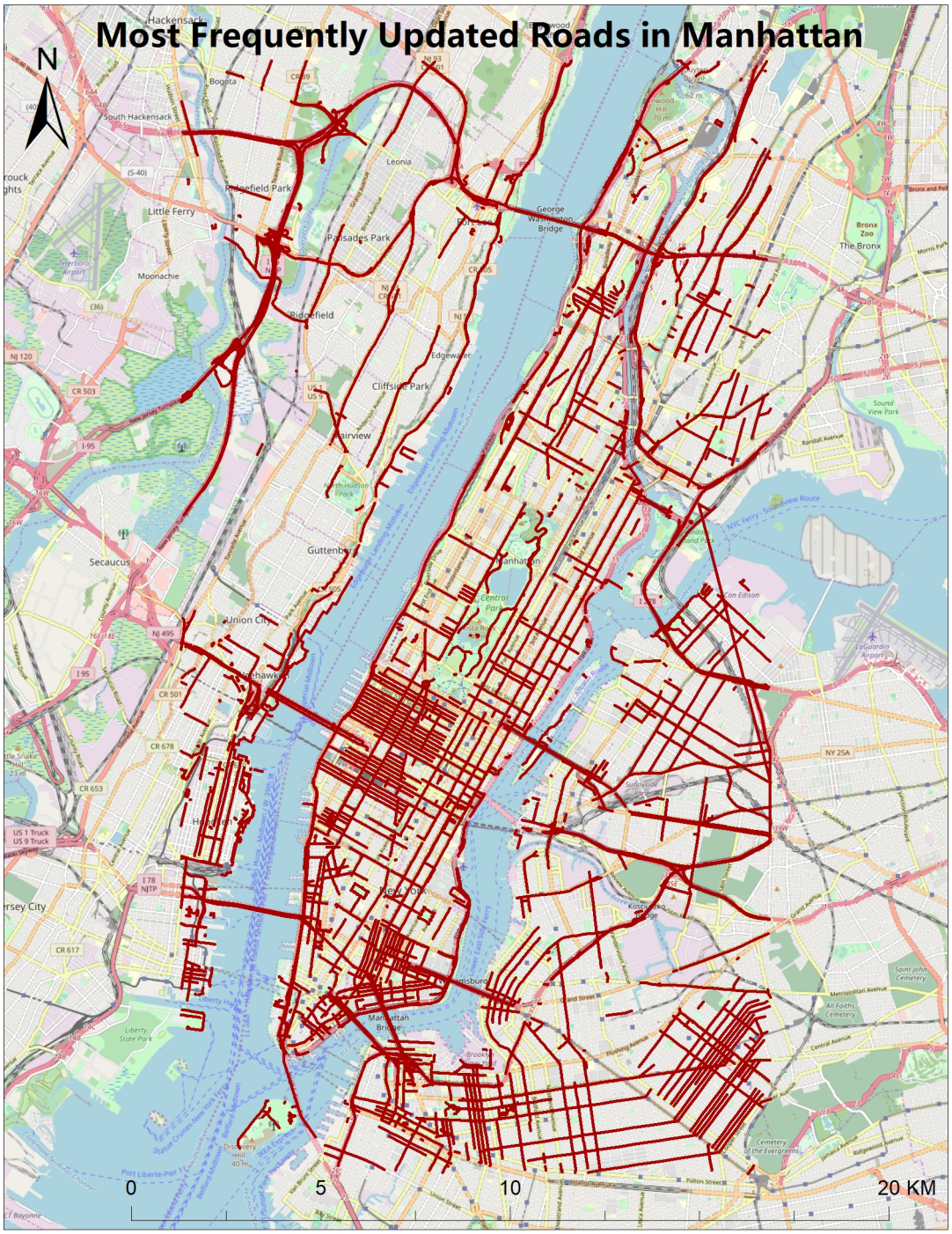


Figure 5. Most Frequently Updated Road in Manhattan

After selecting the most frequently updated buildings and roads, it would be a simple count() function to calculate the number of buildings in each types. Here is the query for counting the number of most frequently updated buildings and all buildings:

```
--BUILDING TYPE--
--TOTAL
SELECT c.type, count(frequency)
FROM(
     SELECT osm_id, b.type,
```

```
            CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                 THEN (max(osm_version) - min (osm_version))
                      /(date_part('day', max(osm_time) - min(osm_time)))
            ELSE NULL
            END AS frequency,
            way
     FROM manhattan_building as a
     JOIN lookup_building as b
     ON a.building_type = b.typecode
     GROUP BY osm_id, b.type, way
     ORDER BY frequency DESC
) c
GROUP BY type
;

--FREQUENT
SELECT type, count(frequency)
FROM(
     SELECT osm_id, b.type,
            CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                 THEN (max(osm_version) - min (osm_version))
                      /(date_part('day', max(osm_time) - min(osm_time)))
            ELSE NULL
            END AS frequency,
            way
     FROM manhattan_building as a
     JOIN lookup_building as b
     ON a.building_type = b.typecode
     GROUP BY osm_id, b.type, way
     ORDER BY frequency DESC
) c
WHERE frequency > (
                    SELECT avg(frequency)
                    FROM(
                         SELECT osm_id, b.type,
                         CASE WHEN date_part('day', max(osm_time) - min(osm_time)) >
 0
                              THEN (max(osm_version) - min (osm_version))
                                   /(date_part('day', max(osm_time) - min(osm_tim
e)))
                         ELSE NULL
                         END AS frequency,
                         way
                         FROM manhattan_building as a
                         JOIN lookup_building as b
                         ON a.building_type = b.typecode
                         GROUP BY osm_id, b.type, way
                         ORDER BY frequency DESC
                         ) d
)
GROUP BY type
;
```

The result is graphed in figure 6 and it shows that other than building of unknown type, civic buildings are the most frequently updated.

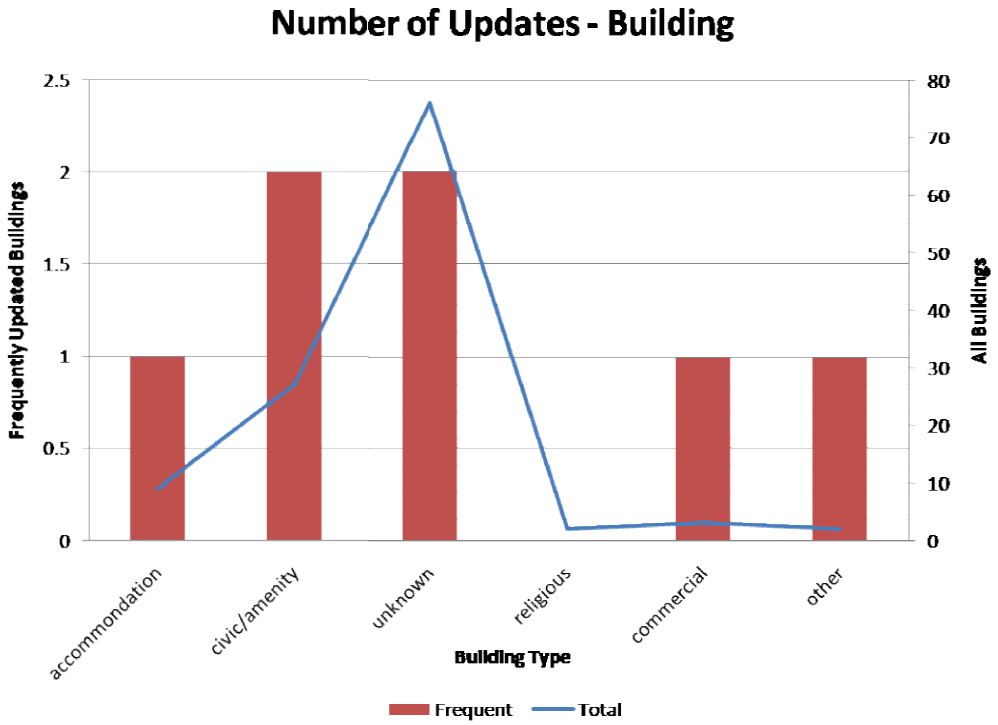


Figure 6. The Type of Most Frequently Updated Buildings

And the same query can be applied for roads:

```
--ROAD TYPE--
--TOTAL
SELECT c.type, count(frequency)
FROM(
     SELECT osm_id, b.type,
            CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                 THEN (max(osm_version) - min (osm_version))
                       /(date_part('day', max(osm_time) - min(osm_time)))
            ELSE NULL
            END AS frequency,
            way
     FROM manhattan_road as a
     JOIN lookup_road as b
     ON a.road_type = b.typecode
     GROUP BY osm_id, b.type, way
     ORDER BY frequency DESC
) c
GROUP BY type
;

--FREQUENT
SELECT type, count(frequency)
FROM(
     SELECT osm_id, b.type,
```

```
            CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                 THEN (max(osm_version) - min (osm_version))
                      /(date_part('day', max(osm_time) - min(osm_time)))
            ELSE NULL
            END AS frequency,
            way
     FROM manhattan_road as a
     JOIN lookup_road as b
     ON a.road_type = b.typecode
     GROUP BY osm_id, b.type, way
     ORDER BY frequency DESC
) c
WHERE frequency > (
                    SELECT avg(frequency)
                    FROM(
                         SELECT osm_id, b.type,
                         CASE WHEN date_part('day', max(osm_time) - min(osm_time)) >
 0
                              THEN (max(osm_version) - min (osm_version))
                                   /(date_part('day', max(osm_time) - min(osm_tim
e)))
                         ELSE NULL
                         END AS frequency,
                         way
                         FROM manhattan_road as a
                         JOIN lookup_road as b
                         ON a.road_type = b.typecode
                         GROUP BY osm_id, b.type, way
                         ORDER BY frequency DESC
                         ) d
)
GROUP BY type
;
```

The result is in figure 7 and it seems that residential, path, and secondary are mostly updated, but they also take up a large proportion in the database.

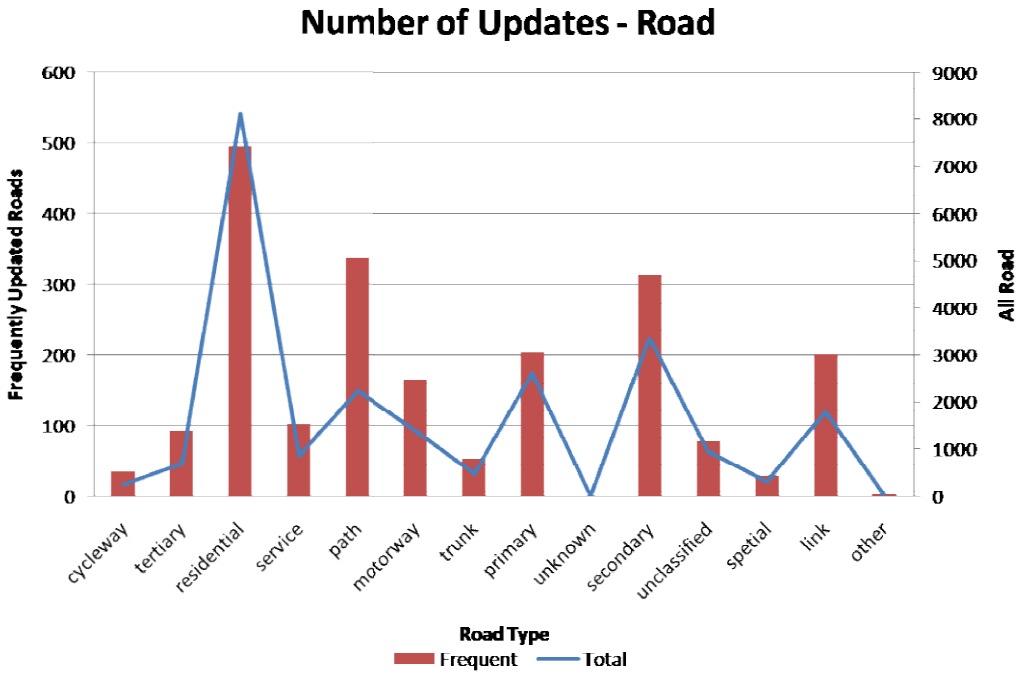


Figure 7. The Type of Most Frequently Updated roads

The range of building is obtained with st_extent(geometry) function. It returns a bounding box containing all records in the table, but not calculated for areas. So I added st_area(geometry) function for calculating the area of this bounding box.

The query returned range = 126146770.590694. Because the data is projected in EPSG 3857 (WGS84 Pseudo-Mercator), the unit is meter. In this case the area of range is 126146770.590694 square meters.

```
--MOST FREQUENT BUILDING RANGE--
SELECT ST_AREA(ST_Extent(way))
FROM(
     SELECT osm_id, b.type,
            CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                 THEN (max(osm_version) - min (osm_version))
                       /(date_part('day', max(osm_time) - min(osm_time)))
            ELSE NULL
            END AS frequency,
            way
     FROM manhattan_building as a
     JOIN lookup_building as b
     ON a.building_type = b.typecode
     GROUP BY osm_id, b.type, way
     ORDER BY frequency DESC
) c
WHERE frequency > (
                    SELECT avg(frequency)
                    FROM(
                         SELECT osm_id, b.type,
                         CASE WHEN date_part('day', max(osm_time) - min(osm_time)) >
 0
```

```
                              THEN (max(osm_version) - min (osm_version))
                                   /(date_part('day', max(osm_time) - min(osm_tim
e)))
                         ELSE NULL
                         END AS frequency,
                         way
                         FROM manhattan_building as a
                         JOIN lookup_building as b
                         ON a.building_type = b.typecode
                         GROUP BY osm_id, b.type, way
                         ORDER BY frequency DESC
                         ) d
)
;
```

The range of the most frequently updated roads are also calculated with this query. It returned the area of range as 458158195.872856 square meters.

```
--MOST FREQUENT ROAD RANGE--
SELECT ST_AREA(ST_Extent(way))
FROM(
     SELECT osm_id, b.type,
            CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                 THEN (max(osm_version) - min (osm_version))
                      /(date_part('day', max(osm_time) - min(osm_time)))
            ELSE NULL
            END AS frequency,
            way
     FROM manhattan_road as a
     JOIN lookup_road as b
     ON a.road_type = b.typecode
     GROUP BY osm_id, b.type, way
     ORDER BY frequency DESC
) c
WHERE frequency > (
                    SELECT avg(frequency)
                    FROM(
                         SELECT osm_id, b.type,
                         CASE WHEN date_part('day', max(osm_time) - min(osm_time)) >
 0
                              THEN (max(osm_version) - min (osm_version))
                                   /(date_part('day', max(osm_time) - min(osm_tim
e)))
                         ELSE NULL
                         END AS frequency,
                         way
                         FROM manhattan_road as a
                         JOIN lookup_road as b
                         ON a.road_type = b.typecode
                         GROUP BY osm_id, b.type, way
                         ORDER BY frequency DESC
                         ) d
)
;
```

Total Updates and The Range of Updates by Contributors

- Is there a difference in the range of updates for buildings or roads for each contributor? Is there a relation between contributors' total update and the range of updates?

Contributors are identified through osm_uid. Total update of a contributor is calculated as the sum of update for buildings and roads.

The query below returns the total update, building update, road update, and range of roads and buildings by contributors.

```
--TOTAL UPDATE BY USER--
SELECT c.osm_uid, building_update, road_update, total_update, building_range, road_
range
FROM(
     --TOTAL UPDATE
     SELECT *, (COALESCE(building_update,0)+COALESCE(road_update,0)) AS total_updat
e
     FROM (
           --building update
           SELECT osm_uid, count(*) AS building_update
           FROM manhattan_building
           GROUP BY osm_uid
          ) AS a
     FULL OUTER JOIN (
           --road update
           SELECT osm_uid, count(*) AS road_update
           FROM manhattan_road
           GROUP BY osm_uid
           ) AS b
     USING (osm_uid)
     ORDER BY total_update DESC
) AS c
FULL OUTER JOIN (
     --BUILDING RANGE BY USER--
     SELECT osm_uid, ST_AREA(ST_Extent(way)) AS building_range
     FROM manhattan_building
     GROUP BY osm_uid
     ORDER BY building_range DESC
) AS d
ON c.osm_uid = d.osm_uid
FULL OUTER JOIN(
     --ROAD RANGE BY USER--
     SELECT osm_uid, ST_AREA(ST_Extent(way)) AS road_range
     FROM manhattan_road
     GROUP BY osm_uid
     ORDER BY road_range DESC
) AS e
ON c.osm_uid = e.osm_uid
;
```

I constructed the query with these steps:

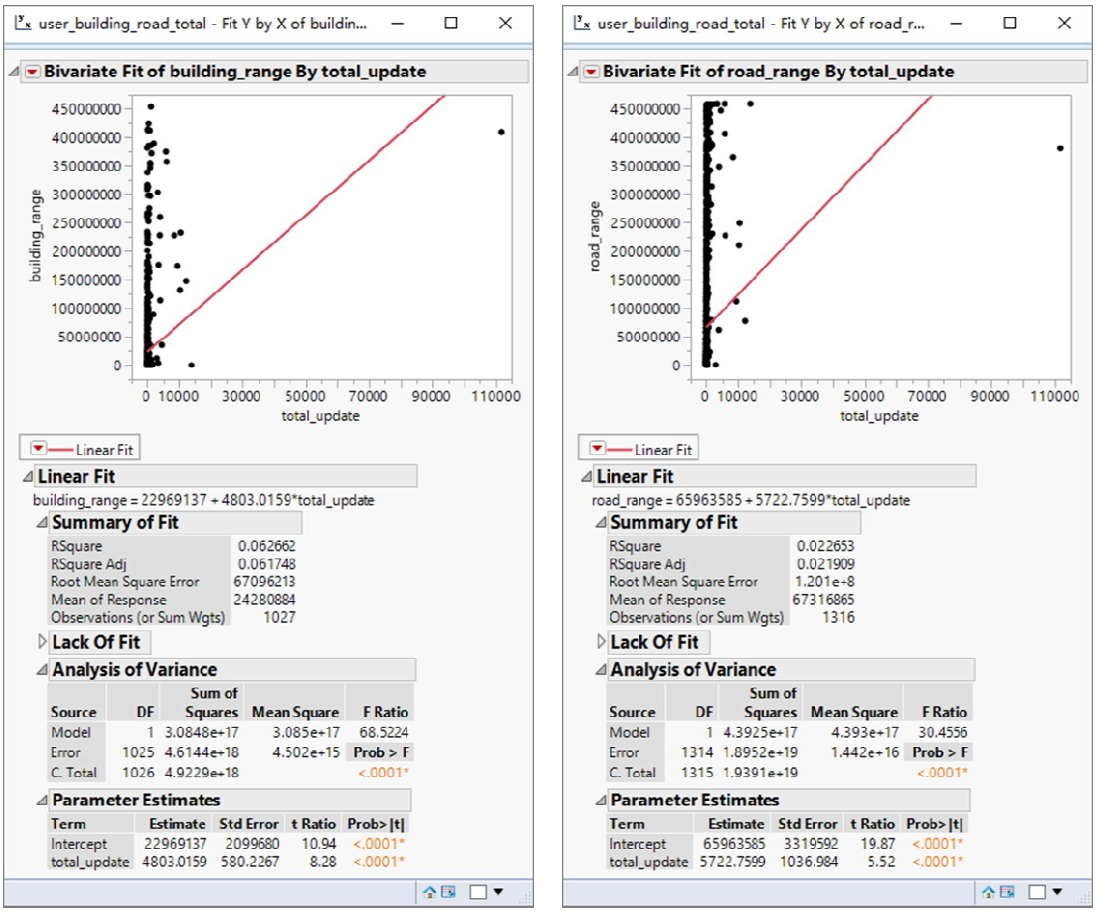
1. count for the frequency of each osm_uid separately from building and road table, as each OSM updates are related to only one contributor in the normalized database.

2. sum up the amount of updates for building and roads as the total update for each contributors.COALESCE(column,0) automatically take null as 0 during calculation. It is required here otherwise total updates of contributors who updated for only road or buildings will be calculated as null.

3. calculate the range of building and roads updated by each user using st_extent(geometry) for getting the bounding box and st_area(geometry) for calculating the area of it.

4. full outer join these tables. I use full outer join here because some users might not participate in updating building or roads.

The regression model of the relations between the range of buildings total updates as well as the range of roads and total updates are shown in figure 8. It suggested that the relations are pretty weak.



Building Range – Total Update **Road Range – Total Update**

Figure 8. Correlation: Range of Building/Road - Total Updates

Frequently Updated Buildings VS. Road type, Frequently Updated Roads VS. Building Type

- What kind of road is the nearest to the most frequently updated buildings? What kind of building the nearest to the most frequently updated roads?

To answer the question about the most updated buildings and what type of roads is closest to it, I am expected to get a table with two columns. One column lists all types of roads and the other calculate the average distance between this type of roads to the most frequently updated buildings.

Here is the query for getting the distance between different types of roads and the most frequently updated buildings.

```
--MOST FREQUENT BUILDING VS ROAD TYPE--
SELECT j.type, AVG(ST_DISTANCE(i.way, j.way))
FROM(
     -- most frequently updated buildings
     SELECT *
     FROM(
          SELECT osm_id, b.type,
                 CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                      THEN (max(osm_version) - min (osm_version))
                            /(date_part('day', max(osm_time) - min(osm_time)))
                 ELSE NULL
                 END AS frequency,
                 way
          FROM manhattan_building as a
          JOIN lookup_building as b
          ON a.building_type = b.typecode
          GROUP BY osm_id, b.type, way
          ORDER BY frequency DESC
     ) c
     WHERE frequency > (
                         SELECT avg(frequency)
                         FROM(
                              SELECT osm_id, b.type,
                              CASE WHEN date_part('day', max(osm_time) - min(osm_ti
me)) > 0
                                   THEN (max(osm_version) - min (osm_version))
                                         /(date_part('day', max(osm_time) - min(osm
_time)))
                              ELSE NULL
                              END AS frequency,
                              way
                              FROM manhattan_building as a
                              JOIN lookup_building as b
                              ON a.building_type = b.typecode
                              GROUP BY osm_id, b.type, way
                              ORDER BY frequency DESC
                              ) d
                        )
) AS i
```

```
JOIN (
       --roads with types
       SELECT *
       FROM manhattan_road AS e
       JOIN lookup_road AS f
       ON e.road_type = f.typecode
) AS j
ON ST_DISTANCE(i.way, j.way) IS NOT NULL
GROUP BY  j.type
;
```

Since there is no relation other than spatial relation between roads and buildings, I joined them based on the geometry column. By joining them on ST_DISTANCE(i.way, j.way) IS NOT NULL, I joined all of the record on the left table to each record on the right table because the statement is always true.

The result for it is shown in figure 9. Although there are not much differences observed, the cycleway and unclassified roads generally have closer distance to buildings.

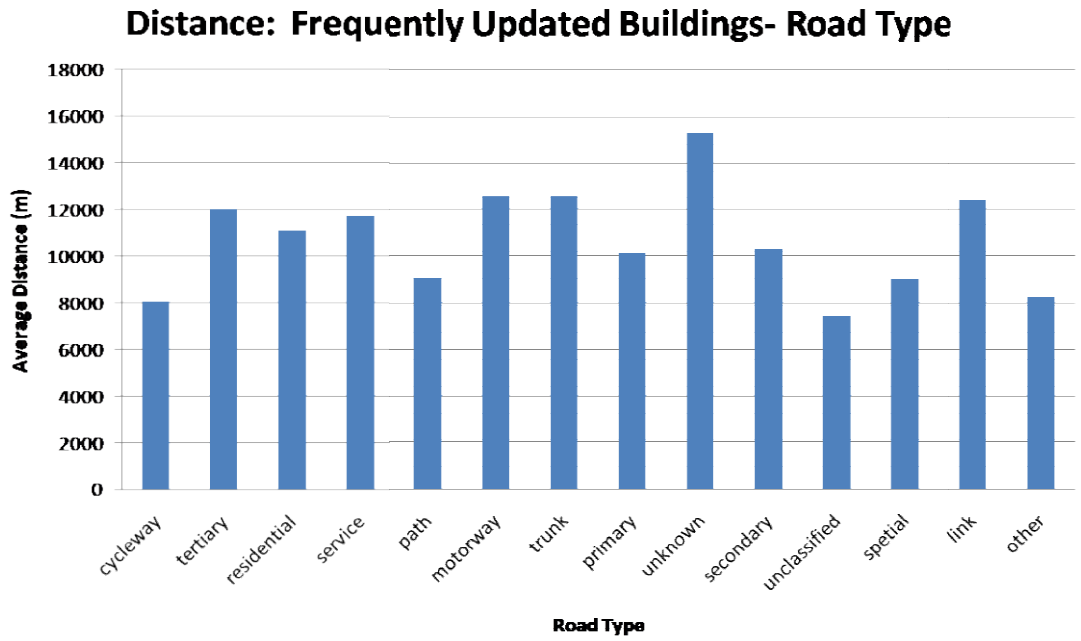


Figure 9. Distance: Frequently Updated Building - Road Type;

Similar methods is also applied for the most frequently updated roads and different types of buildings. But it ran too long so I did some optimization to it. The code is included in Appendix 4

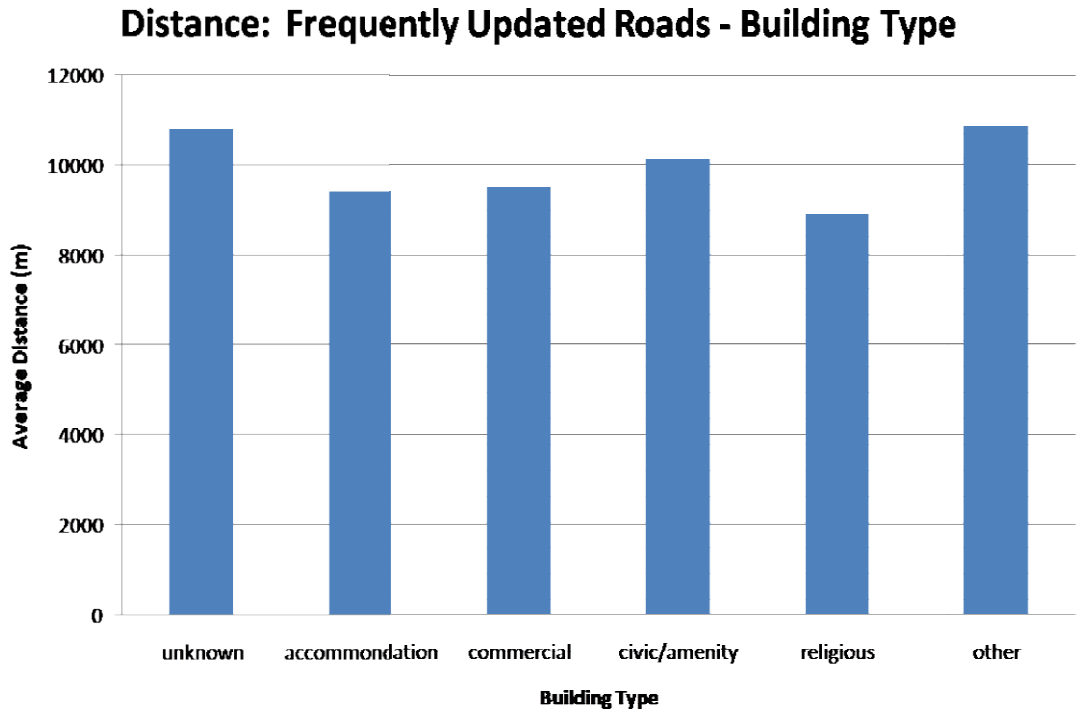


Figure 10. Distance: Frequently Updated Road - building Type;

Appendix 1: Obtaining and Loading Data

Data applied in this study is clipped from the New York data previously clipped from the OSM history dump file downloaded in from: https://planet.openstreetmap.org/planet/full-history/ by the bounding box (N: 40.88 S: 40.68 E: -73.90 W: -74.04).

The file is downloaded in a PBF format with a compressed size of 69 GB at the end of October. It was first clipped by the bounding box with osmconvert in OSGeo-Live.

```
$ osmconvert /media/sf_SHAREFOLDERS_Linux_OSGEO/history-latest.osm.pbf \
> -b=-74.04, 40.68,-73.90,40.88 -o=manhattan_history.pbf
```

Then it was imported with osm2gsql into PostGIS database following these process.

create database and schema. The line below creates a database called osmny, and adds the PostGIS and hstore extensions. The hstore extension is important here because OSM database is not a relational database.

```
$ createdb osmny
$ psql -d osmny -c 'CREATE EXTENSION postgis; CREATE EXTENSION hstore;'
```

Then import data into the database. This code imports the pbf file into the database that we just created.

```
$ osm2pgsql -c -d osmny -C 15000 --hstore --extra-attributes --multi-geometry \
> ~/Desktop/manhattan_history.pbf
```

Appendix 2: Normalization Script

```
set search_path to term_project, public;

--BUILDING--
--CREATE LOOKUP TABLE
CREATE TABLE term_project.lookup_building
(
 typecode int PRIMARY KEY,
 type text
)
;
INSERT INTO term_project.lookup_building (typecode, type)
VALUES (1, 'accommondation'),
       (2, 'commercial'),
       (3, 'civic/amenity'),
       (4, 'religious'),
       (0, 'unknown'),
       (99, 'other')
;

--CREATE BUILDING TABLE
CREATE TABLE term_project.manhattan_building1 AS
SELECT CAST(CONCAT(CAST (osm_id AS text), '000',CAST (osm_version AS text)) AS bigi
nt) AS oid,
       osm_id,
       osm_uid, osm_version, osm_time,
       name,
       CASE --accommondation
            WHEN building = 'accommondation' OR building = 'apartments'
              OR building = 'dormitory' OR building = 'hotel'
              OR building = 'house' OR building = 'hut'
              OR building = 'mansion' OR building = 'motel'
              OR building = 'residential'
            THEN 1
            --commercial
            WHEN building = 'commercial' OR building = 'brewery'
              OR building = 'retail'  OR building = 'cinema'
              OR building = 'food_and_drink' OR building = 'industrial'
              OR building = 'manufacture' OR building = 'office'
              OR building = 'supermarket' OR building = 'theatre'
              OR building = 'warehouse' OR building = 'works'
            THEN 2
            --civic/amenity
            WHEN building = 'amenity' OR building = 'civic'
              OR building = 'college' OR building = 'courthouse'
              OR building = 'fire_station' OR building = 'hospital'
              OR building = 'library' OR building = 'museum'
              OR building = 'public' OR building = 'school'
              OR building = 'stadium' OR building = 'train_station'
              OR building = 'transportation' OR building = 'university'
            THEN 3
            --religious
```

```
            WHEN building = 'religious' OR building = 'cathedral'
              OR building = 'chapel' OR building = 'church'
              OR building = 'convent' OR building = 'mosque'
              OR building = 'synagogue'
            THEN 4
            --unknown
            WHEN building = 'unknown'
            THEN 0
       --others
       ELSE 99
       END AS building_type,
       way
FROM (
      --Assigning building type
      SELECT osm_id,
             osm_uid, osm_version, osm_time,
             name,
             CASE WHEN building != 'yes'
                  THEN building
                  WHEN building = 'yes' AND amenity is not null
                  THEN 'amenity'
                  WHEN building = 'yes' AND amenity is null
                   AND (religion is not null OR denomination is not null)
                  THEN 'religious'
                  WHEN building = 'yes' AND amenity is null
                   AND (landuse = 'commercial' OR landuse = 'industrial'
                        OR landuse = 'retail'
                        OR office is not null OR shop is not null)
                  THEN 'commercial'
                  WHEN building = 'yes' AND amenity is null
                   AND tourism = 'hotel'
                  THEN 'accommondation'
                  WHEN building = 'yes' AND amenity is null
                   AND (leisure is not null OR public_transport is not null)
                  THEN 'amenity'
             ELSE 'unknown'
             END AS building,
             way
      FROM (
            --Select building
            SELECT osm_id, way,
                   CAST(tags -> 'osm_uid' AS bigint) AS osm_uid,
                   CAST(tags -> 'osm_version' AS int) AS osm_version,
                   CAST(tags -> 'osm_timestamp' AS timestamp) AS osm_time,
                   name, building, amenity, landuse,
                   tourism, office, shop,
                   religion, denomination, leisure, public_transport
            FROM planet_osm_polygon
            WHERE building is not null AND building NOT ilike 'no'
           ) AS a
) AS b
;
```

```
--ROAD--
--CREATE LOOKUP TABLE
CREATE TABLE term_project.lookup_road
(
 typecode int PRIMARY KEY,
 type text
)
;
INSERT INTO term_project.lookup_road (typecode, type)
VALUES (1, 'motorway'),
       (2, 'trunk'),
       (3, 'primary'),
       (4, 'secondary'),
       (5, 'tertiary'),
       (6, 'residential'),
       (7, 'service'),
       (8, 'cycleway'),
       (9, 'unclassified'),
       (10, 'link'),
       (20, 'spetial'),
       (30, 'path'),
       (99, 'other'),
       (0, 'unknown')
;
CREATE TABLE term_project.lookup_oneway
(
 code int PRIMARY KEY,
 direction text
)
;
INSERT INTO term_project.lookup_oneway (code, direction)
VALUES (1, 'one way'),
       (2, 'both way'),
       (0, 'unknown')
;

--CREATE ROAD TABLE
CREATE TABLE term_project.manhattan_road_nomerge AS
SELECT CAST(CONCAT(CAST (osm_id AS text), '000',CAST (osm_version AS text)) AS bigi
nt) AS oid,
       osm_id,
       osm_uid, osm_version, osm_time,
       name,
       CASE WHEN oneway = 'yes' OR ONEWAY = '-1' THEN 1
            WHEN oneway = 'no' THEN 2
       ELSE 0
       END AS direction,
       CASE WHEN highway = 'motorway' THEN 1
            WHEN highway = 'trunk' THEN 2
            WHEN highway = 'primary' THEN 3
            WHEN highway = 'secondary' THEN 4
            WHEN highway = 'tertiary' THEN 5
            WHEN highway = 'residential' THEN 6
            WHEN highway = 'service' THEN 7
```

```
            WHEN highway = 'cycleway' THEN 8
            WHEN highway = 'unclassified' THEN 9
            WHEN highway = 'link' THEN 10
            WHEN highway = 'spetial' THEN 20
            WHEN highway = 'path' THEN 30
            WHEN highway = 'other' THEN 99
       ELSE 0
       END AS road_type,
       way
FROM (
      --Assigning road type
      SELECT osm_id,
             osm_uid, osm_version, osm_time,
             name, oneway,
             CASE WHEN highway = 'yes'
                  THEN 'unkown'
                  WHEN highway = 'motorway' OR highway = 'trunk'
                    OR highway = 'primary' OR highway = 'secondary'
                    OR highway = 'tertiary' OR highway = 'unclassified'
                    OR highway = 'residential' OR highway = 'service'
                  THEN highway
                  WHEN highway ILIKE '%link%'
                  THEN 'link'
                  WHEN highway = 'path' OR highway = 'footway'
                    OR highway = 'bridleway' OR highway = 'steps'
                  THEN 'path'
                  WHEN highway = 'living_street' OR highway = 'pedestrian'
                    OR highway = 'track' OR highway = 'bus_guidway'
                    OR highway = 'escape' OR highway = 'raceway'
                    OR highway = 'road'
                  THEN 'spetial'
                  WHEN highway = 'cycleway'
                  THEN 'cycleway'
             ELSE 'other'
             END AS highway,
             way
      FROM (
            --Select road
            SELECT osm_id, way,
                   CAST(tags -> 'osm_uid' AS bigint) AS osm_uid,
                   CAST(tags -> 'osm_version' AS int) AS osm_version,
                   CAST(tags -> 'osm_timestamp' AS timestamp) AS osm_time,
                   name, oneway,highway
            FROM planet_osm_line
            WHERE highway is not null AND highway NOT ilike 'no'
           ) AS a
) AS b
;

--UNION SEPERATED GEOMETRY
CREATE TABLE term_project.manhattan_road AS
SELECT * FROM
(
SELECT f.oid, f.osm_id, f.osm_uid, f.osm_version,
```

```
       f.osm_time, f.name, f.direction, f.road_type,
       ST_UNION(f.way) AS way
FROM (
      SELECT oid, osm_id, osm_uid, osm_version,
             osm_time, name, direction, road_type, way
      FROM term_project.manhattan_road_nomerge
      GROUP BY oid, osm_id, osm_uid, osm_version,
               osm_time, name, direction, road_type, way
      HAVING count(*) >1
      )as f
GROUP BY f.oid, f.osm_id, f.osm_uid, f.osm_version,
         f.osm_time, f.name, f.direction, f.road_type
UNION
SELECT oid, osm_id, osm_uid, osm_version,
       osm_time, name, direction, road_type, way
FROM term_project.manhattan_road_nomerge
GROUP BY oid, osm_id, osm_uid, osm_version,
         osm_time, name, direction, road_type, way
HAVING count(*) = 1
) as b
;


--DROP NOMERGE TABLE
DROP TABLE term_project.manhattan_road_nomerge;



--CONSTRAINS--
--PK
ALTER TABLE term_project.manhattan_building
ADD PRIMARY KEY (oid)
;
ALTER TABLE term_project.manhattan_road
ADD PRIMARY KEY (oid)
;
--FK
ALTER TABLE term_project.manhattan_building
ADD FOREIGN KEY (building_type)
REFERENCES term_project.lookup_building (typecode)
;
ALTER TABLE term_project.manhattan_road
ADD FOREIGN KEY (road_type)
REFERENCES term_project.lookup_road (typecode),
ADD FOREIGN KEY (direction)
REFERENCES term_project.lookup_oneway (code)
;
```

Appendix 3: Optimization

Create Index

```
--CREATE INDEX
CREATE INDEX ON manhattan_building(oid);
CREATE INDEX ON manhattan_building(osm_uid);
CREATE INDEX ON manhattan_building using gist(way);

CREATE INDEX ON manhattan_road(oid);
CREATE INDEX ON manhattan_road(osm_uid);
CREATE INDEX ON manhattan_road using gist(way);
```

Denormalization

```
--OPTIMIZATION--
--TEMP TABLE
CREATE TABLE road_temp AS
SELECT osm_id, way
FROM(
          SELECT osm_id, b.type,
                 CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                      THEN (max(osm_version) - min (osm_version))
                            /(date_part('day', max(osm_time) - min(osm_time)))
                 ELSE NULL
                 END AS frequency,
                 way
          FROM manhattan_road as a
          JOIN lookup_road as b
          ON a.road_type = b.typecode
          GROUP BY osm_id, b.type, way
          ORDER BY frequency DESC
     ) c
     WHERE frequency > (
                          SELECT avg(frequency)
                          FROM(
                               SELECT osm_id, b.type,
                               CASE WHEN date_part('day', max(osm_time) - min(osm_ti
me)) > 0
                                    THEN (max(osm_version) - min (osm_version))
                                          /(date_part('day', max(osm_time) - min(osm
_time)))
                               ELSE NULL
                               END AS frequency,
                               way
                               FROM manhattan_road as a
                               JOIN lookup_road as b
                               ON a.road_type = b.typecode
                               GROUP BY osm_id, b.type, way
                               ORDER BY frequency DESC
                               ) d
                        )
```

```
;

-- BUILDING TEMP TABLE
CREATE TABLE blink_temp AS
SELECT building_type, way
FROM manhattan_building
;

--INDEX
CREATE INDEX ON road_temp using gist(way);
CREATE INDEX ON blink_temp(building_type);
CREATE INDEX ON blink_temp using gist(way);

--JOIN
CREATE TABLE type_temp AS
SELECT b.building_type, AVG(ST_DISTANCE(a.way, b.way))
FROM road_temp AS a
JOIN blink_temp AS b
ON ST_DISTANCE(a.way, b.way) IS NOT NULL
GROUP BY b.building_type
;

--JOIN TYPE
SELECT b.type, a.avg
FROM type_temp AS a
JOIN lookup_building AS b
ON a.building_type = b.typecode
;

--DROP TABLES
DROP TABLE term_project.road_temp;
DROP TABLE term_project.blink_temp;
DROP TABLE term_project.type_temp;
```

Polygon to Point

```
--CREATE BUilding centroid --2.8s
CREATE TABLE building_c_temp AS
SELECT oid, osm_id, building_type, ST_centroid(way) AS way
FROM manhattan_building
;

--ADD SRID FOR CENTROiD-- 1S
ALTER TABLE building_c_temp
ALTER COLUMN WAY TYPE GEOMETRY (Point,3857)
;

--CREATE INDEX-- 9s
CREATE INDEX ON building_c_temp(oid);
CREATE INDEX ON building_c_temp(osm_id);
CREATE INDEX ON building_c_temp(building_type);
CREATE INDEX ON building_c_temp using gist(way);
```

```
--TEMP TABLE 2.6s
CREATE TABLE road_temp AS
SELECT osm_id, way
FROM(
          SELECT osm_id, b.type,
                 CASE WHEN date_part('day', max(osm_time) - min(osm_time)) > 0
                      THEN (max(osm_version) - min (osm_version))
                            /(date_part('day', max(osm_time) - min(osm_time)))
                 ELSE NULL
                 END AS frequency,
                 way
          FROM manhattan_road as a
          JOIN lookup_road as b
          ON a.road_type = b.typecode
          GROUP BY osm_id, b.type, way
          ORDER BY frequency DESC
     ) c
     WHERE frequency > (
                          SELECT avg(frequency)
                          FROM(
                               SELECT osm_id, b.type,
                               CASE WHEN date_part('day', max(osm_time) - min(osm_ti
me)) > 0
                                    THEN (max(osm_version) - min (osm_version))
                                          /(date_part('day', max(osm_time) - min(osm
_time)))
                               ELSE NULL
                               END AS frequency,
                               way
                               FROM manhattan_road as a
                               JOIN lookup_road as b
                               ON a.road_type = b.typecode
                               GROUP BY osm_id, b.type, way
                               ORDER BY frequency DESC
                               ) d
                     )
;

--INDEX 61ms
CREATE INDEX ON road_temp using gist(way);

--JOIN 2h -43min20s
CREATE TABLE froad_btype_temp AS
SELECT b.building_type, AVG(ST_DISTANCE(a.way, b.way))
FROM road_temp AS a
JOIN building_c_temp AS b
ON ST_DISTANCE(a.way, b.way) IS NOT NULL
GROUP BY b.building_type
;

--JOIN TYPE
SELECT b.type, a.avg
FROM type_temp AS a
```

```
JOIN lookup_building AS b
ON a.building_type = b.typecode
;

--DROP TABLES
DROP TABLE term_project.road_temp;
DROP TABLE term_project.building_c_temp;
```

Appendix 4: Data Dictionary

Table: manhattan_building

Column	Type	Constrain	Example	Comment
oid	bigint	PK	38843681000017	osm_id-000-version
osm_id	bigint	/	388436810	
osm_uid	bigint	/	1211481	
osm_version	integer	/	17	
osm_time	timestamp, no time zone	/	2018-06-26 11:21:41	
name	text	/	American Museum of Natural History	
building_type	integer	FK	3	Ref. lookup_road(typecode)
way	geometry(Geometry,3857)	/	/	

Table: manhattan_road

Column	Type	Constrain	Example	Comment
oid	bigint	PK	58847990700010	osm_id-000-version
osm_id	bigint	/	588479907	
osm_uid	bigint	/	5659851	
osm_version	integer	/	10	
osm_time	timestamp, no time zone	/	2018-06-22 08:56:00	
name	text	/	23rd Street	
direction	integer	FK	0	Ref. lookup_road(typecode)
road_type	integer	FK	7	Ref. lookup_oneway(code)
way	geometry(LineString,3857)	/	/	

Table: lookup_building

Column	Type	Constrain	Example
typecode	integer	PK	1
type	text		accommondation

Table: lookup_road

Column	Type	Constrain	Example
typecode	integer	PK	1
type	text		motorway

Table: lookup_oneway

Column	Type	Constrains	Example
code	integer	PK	1
direction	text		one way