A.

a.

- **Query**

```
select color,sum(popn_total)
from nyc_census_blocks as census
join nyc_subway_stations as subways
on ST_DWithin(subways.geom, census.geom,200)
group by color ;
```

- **Explain**

  First, I join nyc_census_blocks and nyc_subway_stations together with ST_DWithin, which can join the census block that matches the rules “lying within 200 meters of one subway station”. Then, I use group by to group the stations that have the same color so that I can use sum(popn_total) to get the total number of populations living in the census blocks that lie within 200 meters of all subway stations which have the same color route.

- **Result**

```
postgis_lab=# select color,sum(popn_total)
postgis_lab-# from nyc_census_blocks as census
postgis_lab-# join nyc_subway_stations as subways
postgis_lab-# on ST_DWithin(subways.geom, census.geom,200)
postgis_lab-# group by color;
       color        |  sum
--------------------+--------
 AIR-BLUE           |   2658
 BLUE               | 241440
 BLUE-BROWN         |   2429
 BLUE-GREY          |   8428
 BLUE-LIME          |   2457
 BLUE-ORANGE        | 100779
 BROWN              | 160809
 BROWN-ORANGE       |  58347
 BROWN-ORANGE-YELLOW |   5198
 BROWN-YELLOW       |  18402
 CLOSED             |   4823
 GREEN              | 385759
 GREEN-ORANGE       |   3553
 GREEN-RED          |  18380
 GREY               |  97588
 GREY-ORANGE        |  11789
 GREY-PURPLE-YELLOW |   1101
 GREY-YELLOW        |   9179
 LIME               |  60507
 MULTI              |  65547
 ORANGE             | 293005
 ORANGE-LIME        |   9766
 ORANGE-YELLOW      |  52397
 PURPLE             | 100519
 PURPLE-YELLOW      |    781
 RED                | 404704
 RED-GREEN          | 213152
 SI-BLUE            |  27098
 YELLOW             | 217501
(29 rows)
```

b.

- **Query**

```
select neigh.name,
sum(census.popn_total) as total_popn,
ST_Area(neigh.geom) as total_area,
1000000*sum(census.popn_total)/ST_Area(neigh.geom) as dense
from nyc_neighborhoods as neigh
join nyc_census_blocks as census
on ST_Contains(neigh.geom, census.geom)
group by neigh.name, neigh.geom
order by dense desc
limit 3 ;
```

- **Explain**

  First, I join nyc_neighborhoods and nyc_census_blocks together with ST_Contains, which can join census blocks that are totally in one neighborhood. Next, I group the rows that have the same neighborhood name and geom. Then, I can use sum(census.popn_total) to get the total population of census blocks in one neighborhood so that I can calculate the density through sum(census.popn_total) divide the neighborhood area. At last, I order them by density with descending rules, so I can get the neighborhoods which have the top 3 high density with limit 3.

- **Result**

```
postgis_lab=# select neigh.name,
postgis_lab-# sum(census.popn_total) as total_popn,
postgis_lab-# ST_Area(neigh.geom) as total_area,
postgis_lab-# 1000000*sum(census.popn_total)/ST_Area(neigh.geom) as dense
postgis_lab-# from nyc_neighborhoods as neigh
postgis_lab-# join nyc_census_blocks as census
postgis_lab-# on ST_Contains(neigh.geom, census.geom)
postgis_lab-# group by neigh.name, neigh.geom
postgis_lab-# order by dense desc
postgis_lab-# limit 3
postgis_lab-# ;
      name        | total_popn |    total_area    |      dense
------------------+------------+------------------+------------------
 North Sutton Area |      15462 | 328194.000196611 | 47112.3786258652
 Upper East Side   |     191051 | 4198725.41579295 | 45502.1419789413
 East Village      |      59734 | 1632116.71718575 | 36599.0981962363
(3 rows)
```

c.

- **Improve query time method**

```
CREATE INDEX geom_index on nyc_neighborhoods using GiST(geom);
CREATE INDEX geom_index on nyc_census_blocks using GiST(geom);
```

- **Explain**

  I create indexes of geom on both tables using GiST method so that we can get spatial data and join **(join nyc_census_blocks as census on ST_Contains(neigh.geom, census.geom))** them faster. The results below show that this reduces the total execution time from 2970.615 ms to 905.073 ms.

- **Before**

```
QUERY PLAN
--------------------------------------------------------------------------------
Limit  (cost=1025316.13..1025316.14 rows=3 width=887) (actual time=2965.444..2969.010 rows=3 loops=1)
  ->  Sort  (cost=1025316.13..1025316.46 rows=129 width=887) (actual time=2965.442..2969.007 rows=3 loops=1)
        Sort Key: ((('1000000'::double precision * sum(census.popn_total)) / st_area(neigh.geom))) DESC
        Sort Method: top-N heapsort  Memory: 27kB
        ->  Finalize GroupAggregate  (cost=1025265.48..1025314.47 rows=129 width=887) (actual time=2951.342..2968.946 rows=119 loops=1)
              Group Key: neigh.name, neigh.geom
              ->  Gather Merge  (cost=1025265.48..1025303.50 rows=258 width=871) (actual time=2951.292..2968.559 rows=357 loops=1)
                    Workers Planned: 2
                    Workers Launched: 2
                    ->  Partial GroupAggregate  (cost=1024265.46..1024273.70 rows=129 width=871) (actual time=2921.175..2933.445 rows=119 loops=3)
                          Group Key: neigh.name, neigh.geom
                          ->  Sort  (cost=1024265.46..1024267.20 rows=695 width=871) (actual time=2921.126..2930.117 rows=9598 loops=3)
                                Sort Key: neigh.name, neigh.geom
                                Sort Method: external merge  Disk: 7944kB
                                Worker 0:  Sort Method: external merge  Disk: 8096kB
                                Worker 1:  Sort Method: external merge  Disk: 8072kB
                                ->  Nested Loop  (cost=0.00..1024232.65 rows=695 width=871) (actual time=0.506..2872.370 rows=9598 loops=3)
                                      Join Filter: ((neigh.geom ~ census.geom) AND _st_contains(neigh.geom, census.geom))
                                      Rows Removed by Join Filter: 1658544
                                      ->  Parallel Seq Scan on nyc_census_blocks census  (cost=0.00..3435.64 rows=16164 width=249) (actual time=0.010..2.672 rows=12931 loops=3)
                                      ->  Seq Scan on nyc_neighborhoods neigh  (cost=0.00..29.29 rows=129 width=863) (actual time=0.000..0.013 rows=129 loops=38794)
Planning Time: 0.298 ms
Execution Time: 2970.615 ms
```

Planning time: 0.298 ms
Execution Time: 2970.615 ms

- **After**

```
QUERY PLAN
--------------------------------------------------------------------------------
Limit  (cost=13102.78..13102.78 rows=3 width=887) (actual time=902.937..902.939 rows=3 loops=1)
  ->  Sort  (cost=13102.78..13103.10 rows=129 width=887) (actual time=902.936..902.937 rows=3 loops=1)
        Sort Key: ((('1000000'::double precision * sum(census.popn_total)) / st_area(neigh.geom))) DESC
        Sort Method: top-N heapsort  Memory: 27kB
        ->  GroupAggregate  (cost=13071.88..13101.11 rows=129 width=887) (actual time=860.606..902.890 rows=119 loops=1)
              Group Key: neigh.name, neigh.geom
              ->  Sort  (cost=13071.88..13076.05 rows=1668 width=871) (actual time=860.467..892.775 rows=28793 loops=1)
                    Sort Key: neigh.name, neigh.geom
                    Sort Method: external merge  Disk: 24112kB
                    ->  Nested Loop  (cost=3.83..12982.60 rows=1668 width=871) (actual time=0.293..780.248 rows=28793 loops=1)
                          ->  Seq Scan on nyc_neighborhoods neigh  (cost=0.00..29.29 rows=129 width=863) (actual time=0.005..0.110 rows=129 loops=1)
                          ->  Bitmap Heap Scan on nyc_census_blocks census  (cost=3.83..100.28 rows=13 width=249) (actual time=0.353..5.999 rows=223 loops=129)
                                Recheck Cond: (neigh.geom ~ geom)
                                Filter: _st_contains(neigh.geom, geom)
                                Rows Removed by Filter: 204
                                Heap Blocks: exact=22415
                                ->  Bitmap Index Scan on geom_index  (cost=0.00..3.83 rows=39 width=0) (actual time=0.070..0.070 rows=427 loops=129)
                                      Index Cond: (neigh.geom ~ geom)
Planning Time: 0.143 ms
Execution Time: 905.073 ms
```

Planning time: 0.143 ms
Execution Time: 905.073 ms

B.

- **Kml on google map**
  https://www.google.com/maps/d/u/0/edit?mid=1Rmm43-KJ5E_IjPePyGV_k8gC81inYBI&usp=sharing

(1)

- **Kml name**
  popu

- **Query**
  select popu.name
  from ne_10m_populated_places as popu
  join ne_10m_railroads as rail
  on ST_DWithin(rail.geom,popu.geom,100)
  group by popu.geom,popu.name
  order by popu.name asc
  limit 10 ;

- **Explain**
  This query is about finding 10 populated places which are 100 meters within any railroads. Populated places consist of multiple points and railroads consist of multiple lines.

- **Result**

```
test=# select popu.name
test-# from ne_10m_populated_places as popu
test-# join ne_10m_railroads as rail
test-# on ST_DWithin(rail.geom,popu.geom,100)
test-# group by popu.geom,popu.name
test-# order by popu.name asc
test-# limit 10
test-# ;
       name
------------------
 'Ataq
 's-Hertogenbosch
 ?????ng H???i
 ????d??
 ????ng H??
 ???ar??bulus
 ??a??ak
 ??acunday
 ??anakkale
 ??ank??r??
(10 rows)
```

(2)

- **Kml name**
  urban-area

- **Query**
  select urban.gid, urban.area_sqkm, count(distinct popu.name)
  from ne_10m_urban_areas as urban
  join ne_10m_populated_places as popu
  on ST_Contains(urban.geom, popu.geom)
  group by urban.gid
  order by urban.area_sqkm desc
  limit 1 ;

- **Explain**
  This query is about finding an urban area which contains any populated places and has the largest area. Populated places consist of multiple points and urban areas consist of multiple polygons. I show the result on google map and find that it's in Tokyo.

- **Result**

```
test=# select urban.gid, urban.area_sqkm, count(distinct popu.name)
test-# from ne_10m_urban_areas as urban
test-# join ne_10m_populated_places as popu
test-# on ST_Contains(urban.geom, popu.geom)
test-# group by urban.gid
test-# order by urban.area_sqkm desc
test-# limit 1
test-# ;
 gid  | area_sqkm | count
------+-----------+-------
 3470 | 18719.989 |     9
(1 row)
```

(3)

- **Kml name**
  urban-count

- **Query**
  select urban.gid, urban.area_sqkm, count(distinct popu.name)
  from ne_10m_urban_areas as urban
  join ne_10m_populated_places as popu
  on ST_Contains(urban.geom, popu.geom)
  group by urban.gid
  order by count(distinct popu.name) desc
  limit 1 ;

- **Explain**
  This query is about finding an urban area which has the most populated places (有最多populated places在的urban). Populated places consist of multiple points and urban areas consist of multiple polygons. I show the result on google map and find that it's in Taiwan. Too amazing!

- **Result**

```
test=# select urban.gid, urban.area_sqkm, count(distinct popu.name)
test-# from ne_10m_urban_areas as urban
test-# join ne_10m_populated_places as popu
test-# on ST_Contains(urban.geom, popu.geom)
test-# group by urban.gid
test-# order by count(distinct popu.name) desc
test-# limit 1
test-# ;
 gid  | area_sqkm | count
------+-----------+-------
 2730 |  1902.946 |    11
(1 row)
```