Helpful Notes

gcloud sql connect cs411-89 --user=root

Show databases;
Use CS411_Group89_PT1;
Show tables;
Describe raw_data;

Create Statements: Part 1-1 and 1-2

CREATE TABLE raw_data (Measurement_ID INT PRIMARY KEY, State_Code INT, County_Code INT, Site_Num INT, Address CHAR(128), State CHAR(32), County CHAR(32), City CHAR(32), Date DATE, NO2_Units CHAR(32), NO2_Mean FLOAT(12,7), NO2_Max_Value INT, NO2_Max_Hour INT, NO2_AQI int, O3_Units CHAR(32), O3_Mean FLOAT(12,7), O3_Max_Value INT, O3_Max_Hour INT, O3_AQI int, SO2_Units CHAR(32), SO2_Mean FLOAT(12,7), SO2_Max_Value INT, SO2_Max_Hour INT, SO2_AQI int, CO_Units CHAR(32), CO_Mean FLOAT(12,7), CO_Max_Value INT, CO_Max_Hour INT, CO_AQI int);

CREATE TABLE State (id INT NOT NULL AUTO_INCREMENT, state_code INT UNIQUE, name CHAR(32) UNIQUE, PRIMARY KEY (id));

CREATE TABLE County (id INT NOT NULL AUTO_INCREMENT, state_id INT, county_code INT, name CHAR(32), PRIMARY KEY (id), FOREIGN KEY (state_id) REFERENCES State(id) ON DELETE CASCADE ON UPDATE CASCADE);

CREATE TABLE City (id INT NOT NULL AUTO_INCREMENT, county_id INT, name CHAR(32), PRIMARY KEY (id), FOREIGN KEY (county_id) REFERENCES County(id) ON DELETE CASCADE ON UPDATE CASCADE);

CREATE TABLE Site(id INT NOT NULL AUTO_INCREMENT, city_id INT, site_num INT, address CHAR(128), PRIMARY KEY(id), FOREIGN KEY (city_id) REFERENCES City(id) ON DELETE CASCADE ON UPDATE CASCADE);

CREATE TABLE Region (region_id INT NOT NULL AUTO_INCREMENT PRIMARY KEY, name CHAR(128) UNIQUE);

CREATE TABLE Site_Region (site_id INT, region_id INT, PRIMARY KEY(site_id, region_ID), FOREIGN KEY (site_ID) REFERENCES Site(id) ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (region_ID) REFERENCES Region(id) ON DELETE CASCADE ON UPDATE CASCADE);

Create table Measurement(id INT NOT NULL AUTO_INCREMENT, site_id INT, compound_id INT, date DATE, parts_per INT, mean FLOAT(12,7), max_value INT, max_hour INT, aqi int, PRIMARY KEY(id), FOREIGN KEY (compound_id) REFERENCES Compound(id), FOREIGN KEY (site_id) REFERENCES Site(id));

CREATE TABLE Compound (id INT NOT NULL AUTO_INCREMENT, name CHAR(16), PRIMARY KEY (id));

CREATE TABLE all_cities (id INT NOT NULL AUTO_INCREMENT, City CHAR(32), County CHAR(32), STATE CHAR(32), PRIMARY KEY (id));

Authors note: The all_cities and raw_data tables are not exactly a part of the design. They are tables designed to import values from buckets containing CSV files. The raw_data contains (some of) the data provided by the course for the project. All_cities contains (some of) a CSV list of all cities in the United States for the purposes of having more than 1000 rows in three of our tables. Cities imported this way have no measurements and no city or county codes. Due to technical difficulties, the complete CSVs were not uploaded properly. The tables as they stand accurately complete the assignment, however the number of rows will expand in the future.

Proof of Table Sizes: Part 1-3

```
mysql> Select Count(*) from Measurement;
+----------+
| Count(*) |
+----------+
|   538300 |
+----------+
1 row in set (0.13 sec)
```

```
mysql> Select Count(*) from County;
+----------+
| Count(*) |
+----------+
|      912 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> select Count(*) from City;
+----------+
| Count(*) |
+----------+
|     8043 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> select Count(*) from raw_data;
+----------+
| Count(*) |
+----------+
|   134575 |
+----------+
1 row in set (1.38 sec)
```

```
mysql> select Count(*) from all_cities;
+----------+
| Count(*) |
+----------+
|     7969 |
+----------+
1 row in set (0.01 sec)
```

Populate Tables: Useful information

This section is included for our own personal use if and when we need to repopulate the tables. It is not exactly a part of the assignment, but I still felt it proper to include them.

INSERT INTO State (state_code, name) SELECT DISTINCT State_Code, State FROM raw_data;

INSERT INTO County (state_id, county_code, name) SELECT DISTINCT id, County_Code, County FROM (Select County_Code, County, State_Code FROM raw_data) as t1 INNER JOIN (select id, state_code FROM State) as t2 ON t1.State_Code = t2.state_code;

INSERT INTO City (county_id, name) SELECT DISTINCT County.id, City FROM County INNER JOIN (State INNER JOIN raw_data on State.state_code = raw_data.State_Code) on County.state_id = State.id and County.county_code = raw_data.County_Code

INSERT INTO Site(city_id, site_num, address) Select distinct City.id, Site_Num, Address from (City inner join County on City.county_id = County.id) inner join State on County.state_id = State.id inner join raw_data on State.state_code = raw_data.State_Code and County.county_code = raw_data.County_Code and City.name = raw_data.City

```
INSERT INTO Measurement (site_id, compound_id, date, parts_per, mean, max_value,
max_hour, aqi)
Select Site.id as site_id, Compound.id as compound_id, raw_data.Date,
IF(raw_data.NO2_Units = "Parts per billion", 1000000000, 1000000), raw_data.NO2_Mean,
raw_data.NO2_Max_Value, raw_data.NO2_Max_Hour, raw_data.NO2_AQI
FROM Site INNER JOIN City on Site.city_id = City.id
        INNER JOIN County on City.county_id = County.id
   INNER JOIN State on County.state_id = State.id
   INNER JOIN raw_data on Site.site_num = raw_data.Site_Num
                                          AND County.county_code =
raw_data.County_Code
                                          AND State.state_code = raw_data.State_Code
        INNER JOIN Compound
WHERE Compound.name = "NO2";
```

(the above query is modified and ran 4 times for each compound)

```
Insert into County(name, state_id) SELECT DISTINCT all_cities.county, State.id
FROM all_cities INNER JOIN State ON all_cities.state = State.name
WHERE NOT EXISTS (select * from County where name = all_cities.county and state_id =
State.id);
```

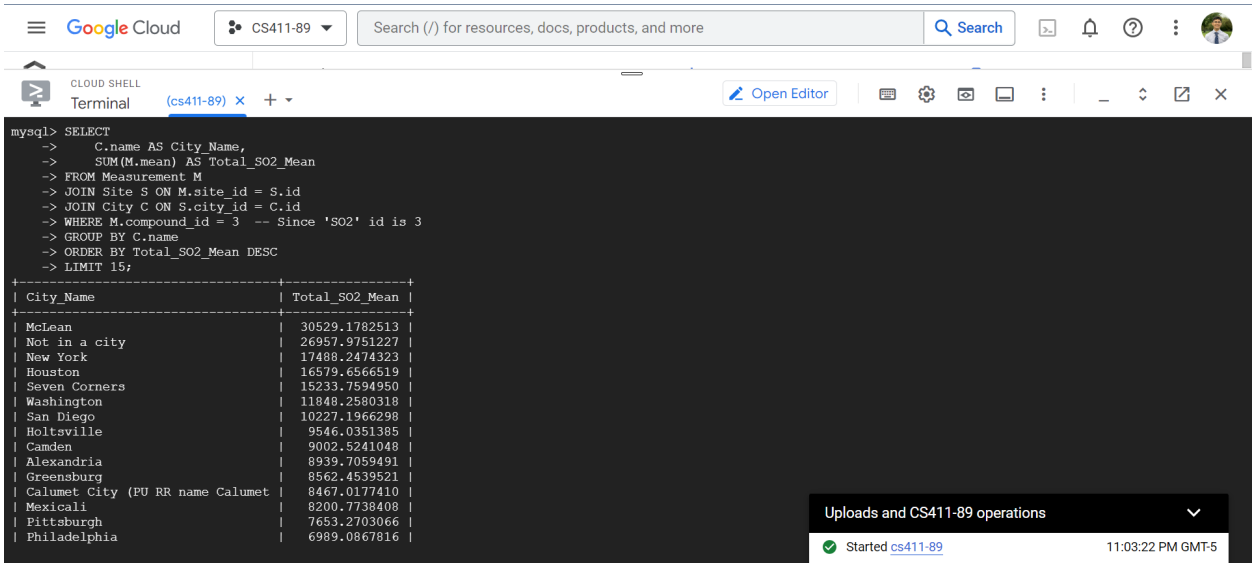
```
Insert into City(name, county_id) SELECT DISTINCT all_cities.city, County.id
FROM all_cities INNER JOIN State ON all_cities.state = State.name
        INNER JOIN County on all_cities.county = County.name AND County.state_id = State.id
WHERE NOT EXISTS (select * from City where name = all_cities.city and county_id =
County.id);
```

Advanced Queries: Part 1-4

Query 1: Finding the top 15 cities with the highest amount mean SO2 measurements

```
SELECT
   C.name AS City_Name,
   SUM(M.mean) AS Total_SO2_Mean
FROM Measurement M
JOIN Site S ON M.site_id = S.id
JOIN City C ON S.city_id = C.id
WHERE M.compound_id = 3
GROUP BY C.name
```

```
ORDER BY Total_SO2_Mean DESC
LIMIT 15;
```



Query 2: Find 15 instances where the AQI values were the highest, indicating potential pollution incidents

```
SELECT
    M.id AS Measurement_ID,
    M.date AS Date,
    C.name AS Compound_Name,
    CI.name AS City_Name,
    M.aqi
FROM Measurement M
JOIN Compound C ON M.compound_id = C.id
JOIN Site S ON M.site_id = S.id
JOIN City CI ON S.city_id = CI.id
ORDER BY M.aqi DESC
LIMIT 15;
```

```
    ->     C.name AS Compound_Name,
    ->     CI.name AS City_Name,
    ->     M.aqi
    -> FROM Measurement M
    -> JOIN Compound C ON M.compound_id = C.id
    -> JOIN Site S ON M.site_id = S.id
    -> JOIN City CI ON S.city_id = CI.id
    -> ORDER BY M.aqi DESC
    -> LIMIT 15;
+---------------+------------+---------------+------------+------+
| Measurement_ID | Date       | Compound_Name | City_Name  | aqi  |
+---------------+------------+---------------+------------+------+
|         311684 | 2007-07-04 | O3            | Mexicali   |  211 |
|         311681 | 2007-07-04 | O3            | Mexicali   |  211 |
|         311682 | 2007-07-04 | O3            | Mexicali   |  211 |
|         311683 | 2007-07-04 | O3            | Mexicali   |  211 |
|         269823 | 2000-06-10 | O3            | Bristol    |  207 |
|         273757 | 2000-06-10 | O3            | Norristown |  207 |
|         273758 | 2000-06-10 | O3            | Norristown |  207 |
|         269822 | 2000-06-10 | O3            | Bristol    |  207 |
|         269821 | 2000-06-10 | O3            | Bristol    |  207 |
|         273760 | 2000-06-10 | O3            | Norristown |  207 |
|         269824 | 2000-06-10 | O3            | Bristol    |  207 |
|         273759 | 2000-06-10 | O3            | Norristown |  207 |
|         264134 | 2000-06-01 | O3            | Charlotte  |  205 |
|         264135 | 2000-06-01 | O3            | Charlotte  |  205 |
|         264133 | 2000-06-01 | O3            | Charlotte  |  205 |
+---------------+------------+---------------+------------+------+
15 rows in set (1.09 sec)

mysql>
```

Part 2: Indexing

Query 1:

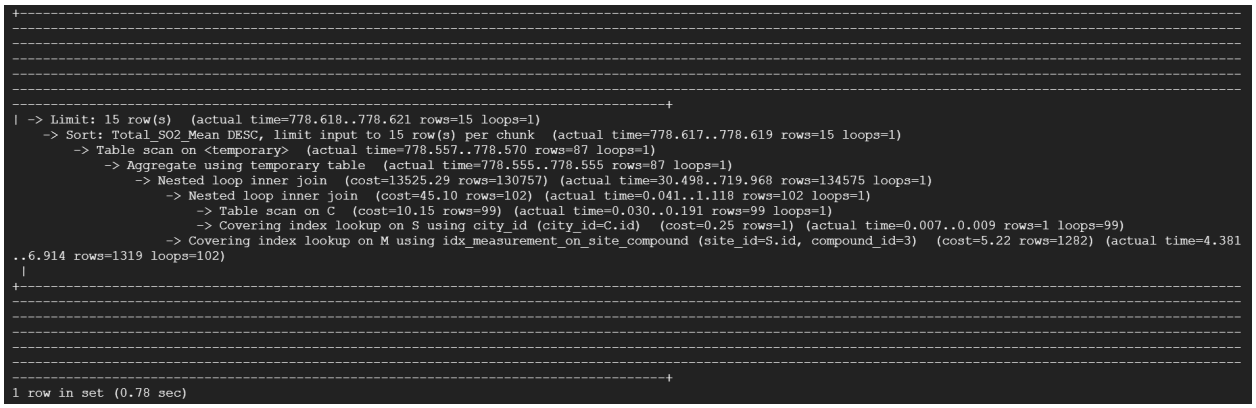
- Original:

```
EXPLAIN ANALYZE SELECT
   C.name AS City_Name,
   SUM(M.mean) AS Total_SO2_Mean
FROM Measurement M
JOIN Site S ON M.site_id = S.id
JOIN City C ON S.city_id = C.id
WHERE M.compound_id = 3
GROUP BY C.name
ORDER BY Total_SO2_Mean DESC
LIMIT 15;
```

```
-> Limit: 15 row(s)  (actual time=857.655..857.660 rows=15 loops=1)
    -> Sort: Total_SO2_Mean DESC, limit input to 15 row(s) per chunk  (actual time=857.654..857.657 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=857.581..857.597 rows=87 loops=1)
            -> Aggregate using temporary table  (actual time=857.580..857.580 rows=87 loops=1)
                -> Nested loop inner join  (cost=192541.71 rows=274995) (actual time=3.752..797.548 rows=134575 loops=1)
                    -> Nested loop inner join  (cost=45.10 rows=102) (actual time=0.059..1.419 rows=102 loops=1)
                        -> Table scan on C  (cost=10.15 rows=99) (actual time=0.040..0.291 rows=99 loops=1)
                        -> Covering index lookup on S using city_id (city_id=C.id)  (cost=0.25 rows=1) (actual time=0.008..0.010 rows=1 loops=99)
                    -> Filter: (M.compound_id = 3)  (cost=1350.66 rows=2696) (actual time=3.847..7.672 rows=1319 loops=102)
                        -> Index lookup on M using site_id (site_id=S.id)  (cost=1350.66 rows=5392) (actual time=0.134..7.224 rows=5277 loops=102)
 |
1 row in set (0.86 sec)
```

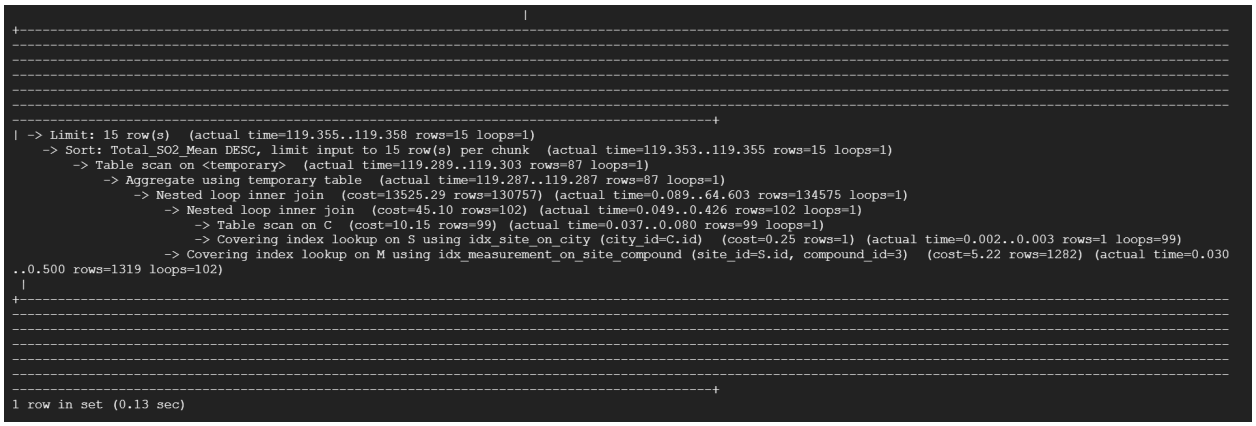
- Design 1: Start with the tables and columns that are directly involved in JOIN and WHERE clauses

CREATE INDEX idx_measurement_on_site_compound ON Measurement(site_id, compound_id, mean);
CREATE INDEX idx_site_on_id_city ON Site(id, city_id);



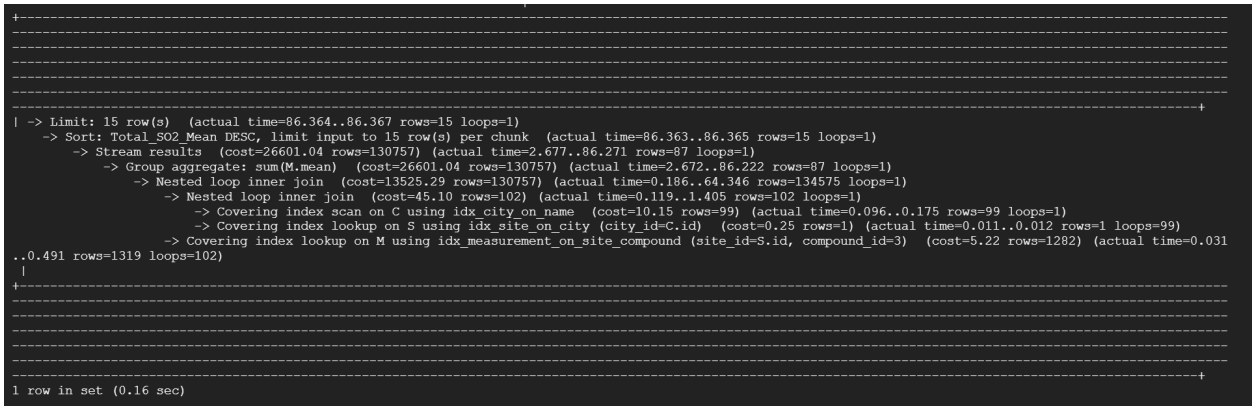
- Design 2: Considering the ORDER BY clause and the JOIN operations.

CREATE INDEX idx_measurement_on_compound_mean ON Measurement(compound_id, mean);
CREATE INDEX idx_site_on_city ON Site(city_id);
CREATE INDEX idx_city_on_id ON City(id);



- Design 3: Focus more on aggregation and sorting

CREATE INDEX idx_measurement_on_compound ON Measurement(compound_id);
CREATE INDEX idx_site_on_city_id ON Site(city_id, id);
CREATE INDEX idx_city_on_name ON City(name);



For the first query, we are deciding to choose Design 2 because it has given us the fastest execution time at 0.13s. This becomes apparent when we see how the actual time to compute the nested loop inner join went from 30 to 0.089 from design 1 to design 2, and how the execution time definitely benefited from that.

Query 2:

- Original:

```
EXPLAIN ANALYZE
SELECT
   M.id AS Measurement_ID,
   M.date AS Date,
   C.name AS Compound_Name,
   CI.name AS City_Name,
   M.aqi
FROM Measurement M
JOIN Compound C ON M.compound_id = C.id
JOIN Site S ON M.site_id = S.id
JOIN City CI ON S.city_id = CI.id
ORDER BY M.aqi DESC
LIMIT 15;
```

```
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=601973.50 rows=15) (actual time=609.389..609.430 rows=15 loops=1)
    -> Nested loop inner join  (cost=601973.50 rows=523030) (actual time=609.387..609.427 rows=15 loops=1)
        -> Nested loop inner join  (cost=418913.00 rows=523030) (actual time=609.378..609.406 rows=15 loops=1)
            -> Nested loop inner join  (cost=235852.50 rows=523030) (actual time=609.354..609.364 rows=15 loops=1)
                -> Sort: M.aqi DESC  (cost=52792.00 rows=523030) (actual time=609.325..609.327 rows=15 loops=1)
                    -> Filter: ((M.compound_id is not null) and (M.site_id is not null))  (cost=52792.00 rows=523030) (actual time=0.066..255.751 rows=538300 lo
ops=1)
                        -> Table scan on M  (cost=52792.00 rows=523030) (actual time=0.064..190.753 rows=538300 loops=1)
                -> Single-row index lookup on C using PRIMARY (id=M.compound_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
            -> Filter: (S.city_id is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
                -> Single-row index lookup on S using PRIMARY (id=M.site_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
        -> Single-row index lookup on CI using PRIMARY (id=S.city_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
 |
+-------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------+
1 row in set (0.62 sec)
```

- Design 1: We're focusing on the primary attributes the query filters and joins on to speed up data retrieval directly from the Measurement and Site tables.

CREATE INDEX idx_measurement_aqi ON Measurement(aqi);
CREATE INDEX idx_measurement_compound ON Measurement(compound_id);
CREATE INDEX idx_site_id ON Site(id);

```
+-------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=392277.01 rows=15) (actual time=0.077..0.120 rows=15 loops=1)
    -> Nested loop inner join  (cost=392277.01 rows=15) (actual time=0.076..0.117 rows=15 loops=1)
        -> Nested loop inner join  (cost=261518.01 rows=15) (actual time=0.071..0.100 rows=15 loops=1)
            -> Nested loop inner join  (cost=130759.01 rows=15) (actual time=0.062..0.074 rows=15 loops=1)
                -> Filter: ((M.compound_id is not null) and (M.site_id is not null))  (cost=0.01 rows=15) (actual time=0.053..0.057 rows=15 loops=1)
                    -> Index scan on M using idx_measurement_aqi (reverse)  (cost=0.01 rows=15) (actual time=0.051..0.053 rows=15 loops=1)
                -> Single-row index lookup on C using PRIMARY (id=M.compound_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
            -> Filter: (S.city_id is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
                -> Single-row index lookup on S using PRIMARY (id=M.site_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
        -> Single-row index lookup on CI using PRIMARY (id=S.city_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
 |
+-------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

- Design 2: We want to optimize the combined filtering and joining operations in the query.

CREATE INDEX idx_measurement_aqi_compound_site ON Measurement(aqi, compound_id, site_id);
CREATE INDEX idx_compound_id ON Compound(id);
CREATE INDEX idx_site_city ON Site(city_id);

```
| -> Limit: 15 row(s)  (cost=392277.01 rows=15) (actual time=0.115..0.182 rows=15 loops=1)
    -> Nested loop inner join  (cost=392277.01 rows=15) (actual time=0.114..0.179 rows=15 loops=1)
        -> Nested loop inner join  (cost=261518.01 rows=15) (actual time=0.103..0.150 rows=15 loops=1)
            -> Nested loop inner join  (cost=130759.01 rows=15) (actual time=0.089..0.107 rows=15 loops=1)
                -> Filter: ((M.compound_id is not null) and (M.site_id is not null))  (cost=0.01 rows=15) (actual time=0.078..0.086 rows=15 loops=1)
                    -> Index scan on M using idx_measurement_aqi_compound_site (reverse)  (cost=0.01 rows=15) (actual time=0.076..0.079 rows=15 loops=1)
                -> Single-row index lookup on C using PRIMARY (id=M.compound_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
            -> Filter: (S.city_id is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
                -> Single-row index lookup on S using PRIMARY (id=M.site_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
        -> Single-row index lookup on CI using PRIMARY (id=S.city_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
 |
1 row in set (0.01 sec)
```

- Design 3: Emphasizes speeding up JOIN operations by indexing the frequently joined attributes together

CREATE INDEX idx_measurement_site_compound ON Measurement(site_id, compound_id);
CREATE INDEX idx_compound_id_3 ON Compound(id);
CREATE INDEX idx_city_id ON City(id);

```
| -> Limit: 15 row(s)  (cost=601973.50 rows=15) (actual time=594.247..594.288 rows=15 loops=1)
    -> Nested loop inner join  (cost=601973.50 rows=523030) (actual time=594.246..594.285 rows=15 loops=1)
        -> Nested loop inner join  (cost=418913.00 rows=523030) (actual time=594.238..594.265 rows=15 loops=1)
            -> Nested loop inner join  (cost=235852.50 rows=523030) (actual time=594.219..594.229 rows=15 loops=1)
                -> Sort: M.aqi DESC  (cost=52792.00 rows=523030) (actual time=594.184..594.186 rows=15 loops=1)
                    -> Filter: ((M.compound_id is not null) and (M.site_id is not null))  (cost=52792.00 rows=523030) (actual time=0.038..258.761 rows=538300 lo
ops=1)
                        -> Table scan on M  (cost=52792.00 rows=523030) (actual time=0.036..191.747 rows=538300 loops=1)
                -> Single-row index lookup on C using PRIMARY (id=M.compound_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
            -> Filter: (S.city_id is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
                -> Single-row index lookup on S using PRIMARY (id=M.site_id)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=15)
        -> Single-row index lookup on CI using PRIMARY (id=S.city_id)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=15)
 |
1 row in set (0.60 sec)
```

For the second query, we are deciding to choose the first design because of its near instantaneous reported speed of ‘0.00s’ and how the design outperformed the others. The idx_measurement_aqi index resulted in a speedier scan of the Measurement table, substantially reducing data retrieval time. The subsequent lookups, such as on the Compound and Site tables using PRIMARY key identifiers, were really efficient with minimal time costs.