EcoVista: Interactive Environmental Insights Map

Team members: ZiHan Li, YiXuan Li, YaTing Pai, Xuanming Zhang

- **Database implementation**
  1. Implementing the database tables locally or on GCP (screenshot)
  2. DDL commands for tables
  3. Inserting at least 1000 rows in the tables (do a count query, screenshot)

Connection to Databases on GCP:

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to trans-aurora-439919-v1.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
yixuan19@cloudshell:~ (trans-aurora-439919-v1)$ gcloud sql connect db-411 --user=root --quiet
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18472
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| EcoVista           |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use EcoVista;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Table 1: AirQuality

```
mysql> CREATE TABLE AirQualityData (
    ->     county_code INT NOT NULL,
    ->     timestamp VARCHAR(7) NOT NULL,
    ->     aqi DECIMAL(5,3),
    ->     PRIMARY KEY (county_code, timestamp),
    ->     FOREIGN KEY (county_code) REFERENCES Location(county_code)
    -> );
Query OK, 0 rows affected (0.21 sec)

mysql> select count(*) from AirQualityData;
+----------+
| count(*) |
+----------+
|     4186 |
+----------+
1 row in set (0.03 sec)

mysql> select * from AirQualityData limit 10;
+-------------+-----------+--------+
| county_code | timestamp | aqi    |
+-------------+-----------+--------+
|        1003 | 2024-01   | 26.172 |
|        1003 | 2024-02   | 34.379 |
|        1003 | 2024-03   | 39.903 |
|        1003 | 2024-04   | 29.773 |
|        1027 | 2024-01   | 23.097 |
|        1027 | 2024-02   | 33.483 |
|        1027 | 2024-03   | 30.677 |
|        1027 | 2024-04   | 36.800 |
|        1049 | 2024-01   | 32.000 |
|        1049 | 2024-02   | 45.000 |
+-------------+-----------+--------+
```

Table 2: COData

```
mysql> CREATE TABLE COData (
    ->     county_code INT NOT NULL,
    ->     timestamp VARCHAR(7) NOT NULL,
    ->     co_measurement DECIMAL(5,3),
    ->     PRIMARY KEY (county_code, timestamp),
    ->     FOREIGN KEY (county_code) REFERENCES Location(county_code)
    -> );
Query OK, 0 rows affected (0.21 sec)

mysql> select * from COData limit 10;
+-------------+-----------+----------------+
| county_code | timestamp | co_measurement |
+-------------+-----------+----------------+
|        1073 | 2023-01   |          0.148 |
|        1073 | 2023-02   |          0.128 |
|        1073 | 2023-03   |          0.161 |
|        1073 | 2023-04   |          0.196 |
|        1073 | 2023-05   |          0.181 |
|        1073 | 2023-06   |          0.198 |
|        1073 | 2023-07   |          0.176 |
|        1073 | 2023-08   |          0.165 |
|        1073 | 2023-09   |          0.169 |
|        1073 | 2023-10   |          0.157 |
+-------------+-----------+----------------+
10 rows in set (0.01 sec)

mysql> select count(*) from COData;
+----------+
| count(*) |
+----------+
|     2463 |
+----------+
1 row in set (0.02 sec)
```

Table 3: DroughtData

```
mysql> CREATE TABLE DroughtData (
    ->     county_code INT NOT NULL,
    ->     timestamp VARCHAR(7) NOT NULL,
    ->     drought_level DECIMAL(5,3),
    ->     PRIMARY KEY (county_code, timestamp),
    ->     FOREIGN KEY (county_code) REFERENCES Location(county_code)
    -> );
Query OK, 0 rows affected (0.26 sec)

mysql> select count(*) from DroughtData;
+----------+
| count(*) |
+----------+
|    19326 |
+----------+
1 row in set (0.03 sec)

mysql> select * from DroughtData limit 10;
+-------------+-----------+---------------+
| county_code | timestamp | drought_level |
+-------------+-----------+---------------+
|        1001 | 2023-12   |        20.000 |
|        1001 | 2024-01   |         9.315 |
|        1001 | 2024-02   |         0.000 |
|        1001 | 2024-03   |         0.000 |
|        1001 | 2024-04   |         0.000 |
|        1001 | 2024-05   |         0.000 |
|        1003 | 2023-12   |         2.066 |
|        1003 | 2024-01   |         4.767 |
|        1003 | 2024-02   |         1.667 |
```

Table 4: NO2 Data

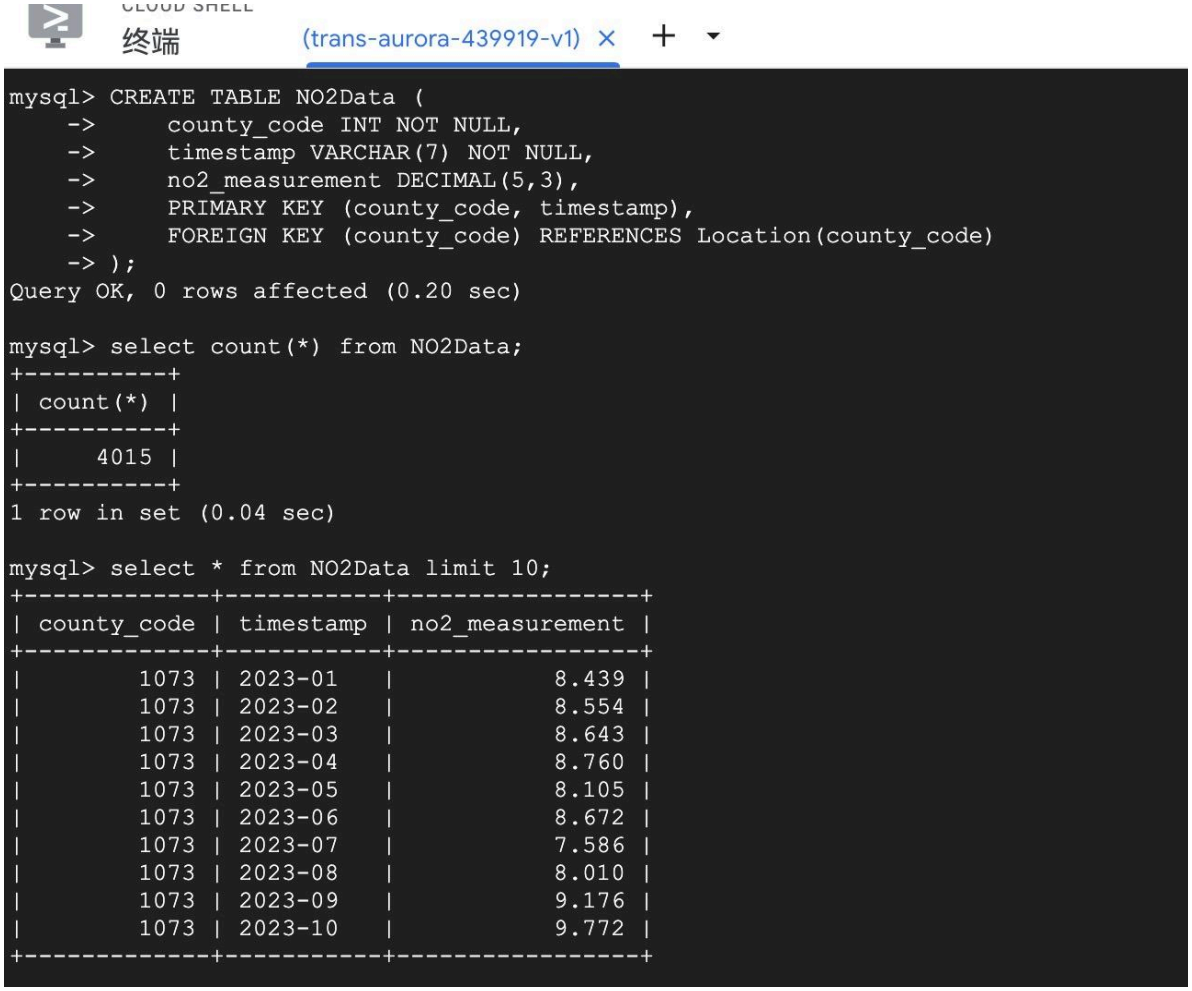


Table 5: UserProfile

```
mysql> CREATE TABLE UserProfile (
    ->     user_id INT PRIMARY KEY,
    ->     username VARCHAR(100),
    ->     county_code INT,
    ->     email VARCHAR(100),
    ->     FOREIGN KEY (county_code) REFERENCES Location(county_code)
    -> );
Query OK, 0 rows affected (0.22 sec)

mysql> select * from UserProfile limit 10;
+---------+----------+-------------+---------------------+
| user_id | username | county_code | email               |
+---------+----------+-------------+---------------------+
|       1 | amjkstvr |       15007 | amjkstvr@gmail.com  |
|       2 | vhsubzum |       48507 | vhsubzum@yahoo.com  |
|       3 | yngrmskk |       48273 | yngrmskk@yahoo.com  |
|       4 | hskxyhfk |       29199 | hskxyhfk@yahoo.com  |
|       5 | siyfrost |       13185 | siyfrost@yahoo.com  |
|       6 | asnqygqe |       31033 | asnqygqe@hotmail.com |
|       7 | glrbkimf |       21149 | glrbkimf@yahoo.com  |
|       8 | eqlmshjs |       42117 | eqlmshjs@hotmail.com |
|       9 | bbpxtcyo |       13045 | bbpxtcyo@gmail.com  |
|      10 | myhswdns |       29111 | myhswdns@yahoo.com  |
+---------+----------+-------------+---------------------+
10 rows in set (0.01 sec)
```

• Queries & Indexing

1. Advanced Queries (1)

Make a ranked list for all counties within a time period (e.g from 2024-01 to 2024-06) while also count the number of users in each county.

```sql
SELECT L.county_name,
       L.state,
       AVG(A.aqi) AS avg_aqi,
       COUNT(U.user_id) AS user_count,
       RANK() OVER (ORDER BY AVG(A.aqi) ASC) AS aqi_rank
FROM AirQualityData A
JOIN Location L ON A.county_code = L.county_code
LEFT JOIN UserProfile U ON A.county_code = U.county_code
WHERE A.timestamp BETWEEN '2024-01' AND '2024-06'
GROUP BY L.county_name, L.state
ORDER BY aqi_rank
LIMIT 15;
```

```
+---------------+-------+------------+------------+----------+
| county_name   | state | avg_aqi    | user_count | aqi_rank |
+---------------+-------+------------+------------+----------+
| Uinta         | WY    |  4.6113333 |          0 |        1 |
| Hopewell City | VA    |  5.6466000 |          0 |        2 |
| Caguas        | PR    |  6.6945000 |          6 |        3 |
| Karnes        | TX    |  7.9380000 |          0 |        4 |
| Carroll       | VA    |  8.3934000 |          5 |        5 |
| Hughes        | SD    |  8.7156667 |          0 |        6 |
| Beaverhead    | MT    |  9.1263333 |          0 |        7 |
| Carbon        | WY    |  9.5390000 |          3 |        8 |
| Washakie      | WY    |  9.6012000 |          5 |        9 |
| Wilson        | TX    | 10.4770000 |          0 |       10 |
| Elko          | NV    | 10.7463333 |          0 |       11 |
| Columbiana    | OH    | 11.6340000 |          6 |       12 |
| Navajo        | AZ    | 11.8206667 |          6 |       13 |
| Park          | WY    | 12.4235000 |          6 |       14 |
| Fremont       | CO    | 12.7574000 |          0 |       15 |
+---------------+-------+------------+------------+----------+
15 rows in set (0.03 sec)
```

- **Indexing Analysis**

Before:

```
| -> Limit: 15 row(s)  (actual time=23.234..23.236 rows=15 loops=1)
    -> Sort: aqi_rank  (actual time=23.232..23.234 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=22.945..23.076 rows=943 loops=1)
            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=22.943..22.943 rows=943 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY avg(A.aqi) )  (actual time=22.117..22.783 rows=943 loops=1)
                    -> Sort: avg_aqi  (actual time=22.105..22.190 rows=943 loops=1)
                        -> Table scan on <temporary>  (actual time=21.309..21.585 rows=943 loops=1)
                            -> Aggregate using temporary table  (actual time=21.305..21.305 rows=943 loops=1)
                                -> Nested loop left join  (cost=758.70 rows=583) (actual time=0.135..14.789 rows=4434 loops=1)
                                    -> Nested loop inner join  (cost=584.12 rows=465) (actual time=0.110..5.865 rows=4071 loops=1)
                                        -> Filter: (A.`timestamp` between '2024-01' and '2024-06')  (cost=421.35 rows=465) (actual time=0.087..3.027 rows=4071 loops=1)
                                            -> Table scan on A  (cost=421.35 rows=4186) (actual time=0.076..1.520 rows=4186 loops=1)
                                        -> Single-row index lookup on L using PRIMARY (county_code=A.county_code)  (cost=0.25 rows=1) (actual time=0.000..0.001 rows=1 loops=4071)
                                    -> Covering index lookup on U using idx_user_profile_county_code (county_code=A.county_code)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0
loops=4071)
```

CREATE INDEX idx_airquality_timestamp ON AirQualityData(timestamp);

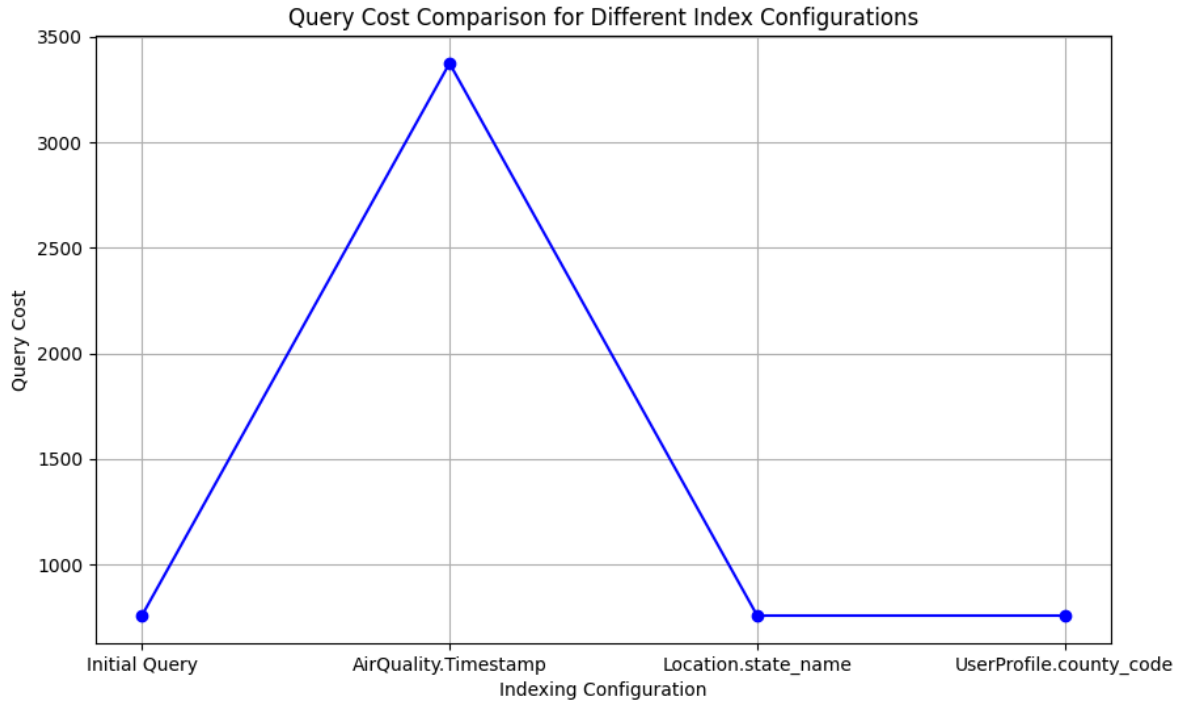
```
| -> Limit: 15 row(s)  (actual time=24.468..24.470 rows=15 loops=1)
    -> Sort: aqi_rank  (actual time=24.466..24.468 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=24.109..24.258 rows=943 loops=1)
            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=24.104..24.104 rows=943 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY avg(A.aqi) )  (actual time=22.995..23.829 rows=943 loops=1)
                    -> Sort: avg_aqi  (actual time=22.965..23.106 rows=943 loops=1)
                        -> Table scan on <temporary>  (actual time=21.667..22.116 rows=943 loops=1)
                            -> Aggregate using temporary table  (actual time=21.663..21.663 rows=943 loops=1)
                                -> Nested loop left join  (cost=3374.38 rows=5102) (actual time=0.155..15.304 rows=4434 loops=1)
                                    -> Nested loop inner join  (cost=1846.20 rows=4071) (actual time=0.135..6.106 rows=4071 loops=1)
                                        -> Filter: (A.`timestamp` between '2024-01' and '2024-06')  (cost=421.35 rows=4071) (actual time=0.116..3.275 rows=4071 loops=1)
                                            -> Table scan on A  (cost=421.35 rows=4186) (actual time=0.111..1.656 rows=4186 loops=1)
                                        -> Single-row index lookup on L using PRIMARY (county_code=A.county_code)  (cost=0.25 rows=1) (actual time=0.000..0.001 rows=1 loops=4071)
                                    -> Covering index lookup on U using idx_user_profile_county_code (county_code=A.county_code)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0
loops=4071)
```

CREATE INDEX idx_location_state_county_name ON Location(state, county_name);

```
| -> Limit: 15 row(s)  (actual time=20.920..20.923 rows=15 loops=1)
    -> Sort: aqi_rank  (actual time=20.920..20.921 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=20.618..20.772 rows=943 loops=1)
            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=20.617..20.617 rows=943 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY avg(A.aqi) )  (actual time=19.827..20.485 rows=943 loops=1)
                    -> Sort: avg_aqi  (actual time=19.815..19.893 rows=943 loops=1)
                        -> Table scan on <temporary>  (actual time=19.078..19.299 rows=943 loops=1)
                            -> Aggregate using temporary table  (actual time=19.074..19.074 rows=943 loops=1)
                                -> Nested loop left join  (cost=758.70 rows=583) (actual time=0.051..13.425 rows=4434 loops=1)
                                    -> Nested loop inner join  (cost=584.12 rows=465) (actual time=0.041..5.234 rows=4071 loops=1)
                                        -> Filter: (A.`timestamp` between '2024-01' and '2024-06')  (cost=421.35 rows=465) (actual time=0.031..2.793 rows=4071 loops=1)
                                            -> Table scan on A  (cost=421.35 rows=4186) (actual time=0.029..1.347 rows=4186 loops=1)
                                        -> Single-row index lookup on L using PRIMARY (county_code=A.county_code)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=4071)
                                    -> Covering index lookup on U using idx_user_profile_county_code (county_code=A.county_code)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0
loops=4071)
 |
```

CREATE INDEX idx_userprofile_county ON UserProfile(county_code);

```
| -> Limit: 15 row(s)  (actual time=22.072..22.074 rows=15 loops=1)
    -> Sort: aqi_rank  (actual time=22.071..22.072 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=21.795..21.922 rows=943 loops=1)
            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=21.794..21.794 rows=943 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY avg(A.aqi) )   (actual time=20.999..21.659 rows=943 loops=1)
                    -> Sort: avg_aqi  (actual time=20.990..21.079 rows=943 loops=1)
                        -> Table scan on <temporary>  (actual time=20.304..20.495 rows=943 loops=1)
                            -> Aggregate using temporary table  (actual time=20.301..20.301 rows=943 loops=1)
                                -> Nested loop left join  (cost=758.70 rows=583) (actual time=0.099..14.342 rows=4434 loops=1)
                                    -> Nested loop inner join  (cost=584.12 rows=465) (actual time=0.070..5.574 rows=4071 loops=1)
                                        -> Filter: (A.`timestamp` between '2024-01' and '2024-06')  (cost=421.35 rows=465) (actual time=0.058..3.026 rows=4071 loops=1)
                                            -> Table scan on A  (cost=421.35 rows=4186) (actual time=0.055..1.492 rows=4186 loops=1)
                                        -> Single-row index lookup on L using PRIMARY (county_code=A.county_code)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=4071)
                                    -> Covering index lookup on U using idx_user_profile_county_code (county_code=A.county_code)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0
loops=4071)
```



Analysis on Query 1:

In our indexing analysis, we observed no significant cost differences when using indexes on Initial Query, Location.county_name&state, and UserProfile.county_code. However, adding an index on AirQualityData.timestamp unexpectedly increased the query cost. Here's a detailed explanation for these observations.

For the Initial Query baseline cost and indexes on Location.county_name and UserProfile.county_code, there were no notable performance improvements. This lack of difference is likely because MySQL's optimizer, when working with smaller datasets, often chooses a full table scan over using indexes. The full scan can sometimes be more efficient, as it avoids the overhead associated with random I/O involved in index lookups, especially on smaller data volumes where the optimizer doesn't see significant benefit in using these indexes.

Additionally, the use of the timestamp index may have introduced extra I/O costs due to MySQL needing to access additional table data beyond what is stored in the index. When a query requires columns not included in the index, MySQL must perform a "table lookup" or "bookmark lookup" to retrieve the remaining data, which adds to the query's cost. In this case, adding an index on timestamp likely triggered extra disk I/O, resulting in an overall increase in query cost.

In conclusion, our final indexing strategy excludes an index on AirQualityData.timestamp due to its added cost, while retaining indexes on Location.county_name and UserProfile.county_code. Although these latter indexes didn't yield noticeable cost improvements, they could become beneficial as the dataset grows, supporting more efficient joins without adversely impacting smaller data operations.

2. Advanced Queries (2)

Analyze and compare environmental metrics (like air quality, CO, NO_2, and drought levels) across different states, along with tracking user engagement (number of users) within each state.

```
SELECT L.state,
       AVG(AQD.aqi) AS avg_aqi,
       AVG(CO.co_measurement) AS avg_co,
       AVG(N.no2_measurement) AS avg_no2,
       AVG(D.drought_level) AS avg_drought,
       COUNT(U.user_id) AS user_count,
       RANK() OVER (ORDER BY AVG(AQD.aqi) DESC) AS aqi_rank,
       RANK() OVER (ORDER BY AVG(CO.co_measurement) DESC) AS co_rank,
```

```
       RANK() OVER (ORDER BY AVG(N.no2_measurement) DESC) AS no2_rank,
       RANK() OVER (ORDER BY AVG(D.drought_level) DESC) AS drought_rank
FROM Location L
LEFT JOIN AirQualityData AQD ON L.county_code = AQD.county_code
LEFT JOIN COData CO ON L.county_code = CO.county_code
LEFT JOIN NO2Data N ON L.county_code = N.county_code
LEFT JOIN DroughtData D ON L.county_code = D.county_code
LEFT JOIN UserProfile U ON L.county_code = U.county_code
GROUP BY L.state
ORDER BY L.state
LIMIT 15;
```

state	avg_aqi	avg_co	avg_no2	avg_drought	user_count	aqi_rank	co_rank	no2_rank	drought_rank
AK	37.9355036	0.3453750	NULL	0.1002256	48	29	6	50	44
AL	40.6115651	0.1736667	9.1378333	12.4889353	336	23	45	17	7
AR	44.9686919	0.3070526	2.8085276	5.5839357	390	13	9	45	19
AZ	59.3985206	0.2595263	11.6113889	14.9733830	14718	1	24	6	6
CA	44.1669001	0.2913991	8.9031496	0.1075216	69558	15	16	18	43
CO	45.6080078	0.2360727	10.1937455	4.3425914	43098	10	31	9	22
Country Of Mexico	56.6751667	NULL	NULL	NULL	0	2	52	50	53
CT	31.4043806	0.2937137	10.9236275	0.0000000	11838	42	13	7	48
DE	38.2239444	NULL	10.0821818	0.0137778	396	28	52	10	46
District Of Columbia	40.5205714	0.3304211	10.4002222	0.0000000	43092	24	7	8	48
FL	43.1917987	0.2777248	6.9872539	5.7312689	642	16	20	30	18
GA	42.4592710	0.4498947	13.9397059	4.9046995	882	18	1	3	21
HI	28.1009236	0.1156875	3.2760000	2.3063576	6	49	49	44	26
IA	42.8166029	0.2040500	4.8059118	10.7811833	1248	17	38	38	9
ID	31.1181984	0.1866250	8.8511333	0.7651684	252	43	42	19	35

15 rows in set (9.94 sec)

- **Indexing Analysis**

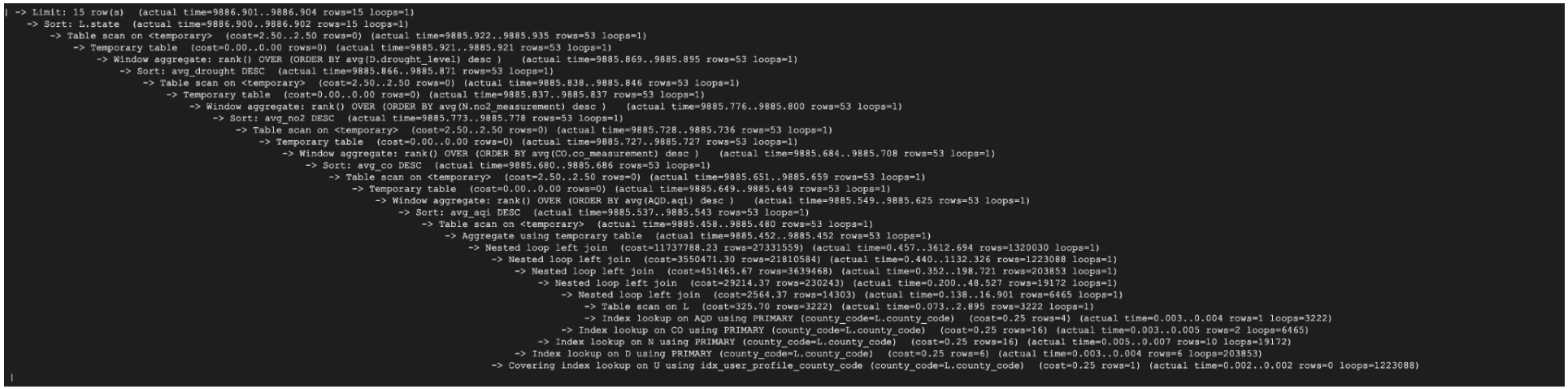
Before:

```
| -> Limit: 15 row(s)  (actual time=9009.583..9009.586 rows=15 loops=1)
    -> Sort: L.state  (actual time=9009.582..9009.584 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=9008.114..9008.128 rows=53 loops=1)
            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=9008.113..9008.113 rows=53 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY avg(D.drought_level) desc )   (actual time=9008.066..9008.092 rows=53 loops=1)
                    -> Sort: avg_drought DESC  (actual time=9008.063..9008.068 rows=53 loops=1)
                        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=9008.035..9008.043 rows=53 loops=1)
                            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=9008.034..9008.034 rows=53 loops=1)
                                -> Window aggregate: rank() OVER (ORDER BY avg(N.no2_measurement) desc )   (actual time=9007.992..9008.016 rows=53 loops=1)
                                    -> Sort: avg_no2 DESC  (actual time=9007.989..9007.994 rows=53 loops=1)
                                        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=9007.962..9007.971 rows=53 loops=1)
                                            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=9007.962..9007.962 rows=53 loops=1)
                                                -> Window aggregate: rank() OVER (ORDER BY avg(CO.co_measurement) desc )   (actual time=9007.907..9007.932 rows=53 loops=1)
                                                    -> Sort: avg_co DESC  (actual time=9007.903..9007.909 rows=53 loops=1)
                                                        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=9007.873..9007.881 rows=53 loops=1)
                                                            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=9007.872..9007.872 rows=53 loops=1)
                                                                -> Window aggregate: rank() OVER (ORDER BY avg(AQD.aqi) desc )   (actual time=9007.755..9007.843 rows=53 loops=1)
                                                                    -> Sort: avg_aqi DESC  (actual time=9007.744..9007.750 rows=53 loops=1)
                                                                        -> Table scan on <temporary>  (actual time=9007.647..9007.669 rows=53 loops=1)
                                                                            -> Aggregate using temporary table  (actual time=9007.640..9007.640 rows=53 loops=1)
                                                                                -> Nested loop left join  (cost=11737788.23 rows=27331559) (actual time=0.113..3253.869 rows=1320030 loops=1)
                                                                                    -> Nested loop left join  (cost=3550471.30 rows=21810584) (actual time=0.104..1000.600 rows=1223088 loops=1)
                                                                                        -> Nested loop left join  (cost=451465.67 rows=3639468) (actual time=0.095..166.634 rows=203853 loops=1)
                                                                                            -> Nested loop left join  (cost=29214.37 rows=230243) (actual time=0.079..39.568 rows=19172 loops=1)
                                                                                                -> Nested loop left join  (cost=2564.37 rows=14303) (actual time=0.071..13.791 rows=6465 loops=1)
                                                                                                    -> Table scan on L  (cost=325.70 rows=3222) (actual time=0.053..2.090 rows=3222 loops=1)
                                                                                                    -> Index lookup on AQD using PRIMARY (county_code=L.county_code)  (cost=0.25 rows=4) (actual time=0.003..0.003 rows=1 loops=3222)
                                                                                                -> Index lookup on CO using PRIMARY (county_code=L.county_code)  (cost=0.25 rows=16) (actual time=0.003..0.004 rows=2 loops=6465)
                                                                                            -> Index lookup on N using PRIMARY (county_code=L.county_code)  (cost=0.25 rows=16) (actual time=0.004..0.006 rows=10 loops=19172)
                                                                                        -> Index lookup on D using PRIMARY (county_code=L.county_code)  (cost=0.25 rows=6) (actual time=0.002..0.004 rows=6 loops=203853)
                                                                                    -> Covering index lookup on U using idx_user_profile_county_code (county_code=L.county_code)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0 loops=12230
88)
 |
```

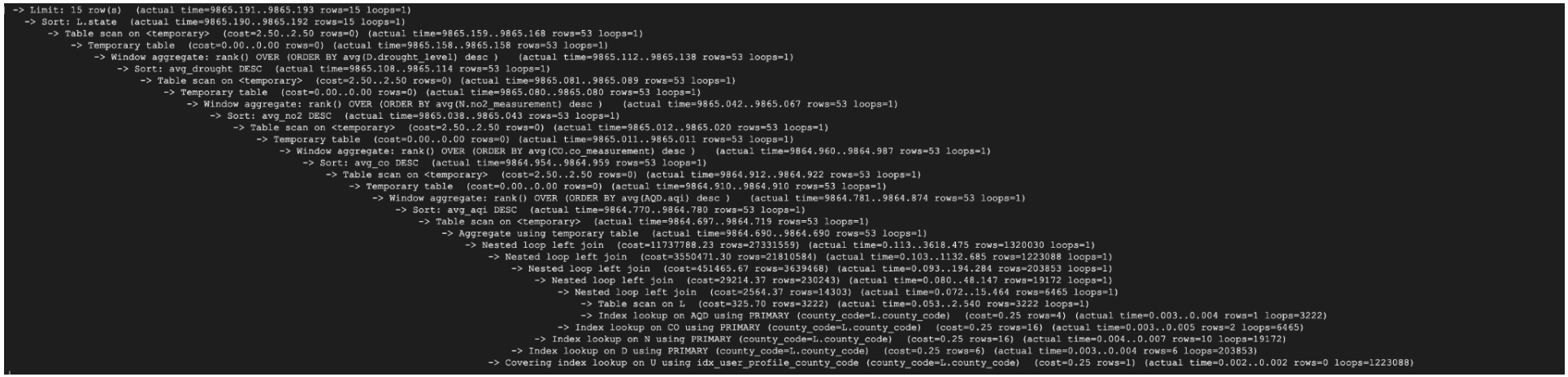
CREATE INDEX idx_user_county ON UserProfile(county_code, user_id);

```
| -> Limit: 15 row(s)  (actual time=9455.939..9455.942 rows=15 loops=1)
    -> Sort: L.state  (actual time=9455.938..9455.940 rows=15 loops=1)
        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=9455.903..9455.911 rows=53 loops=1)
            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=9455.902..9455.902 rows=53 loops=1)
                -> Window aggregate: rank() OVER (ORDER BY avg(D.drought_level) desc )   (actual time=9455.853..9455.878 rows=53 loops=1)
                    -> Sort: avg_drought DESC  (actual time=9455.849..9455.854 rows=53 loops=1)
                        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=9455.819..9455.827 rows=53 loops=1)
                            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=9455.818..9455.818 rows=53 loops=1)
                                -> Window aggregate: rank() OVER (ORDER BY avg(N.no2_measurement) desc )   (actual time=9455.755..9455.780 rows=53 loops=1)
                                    -> Sort: avg_no2 DESC  (actual time=9455.750..9455.756 rows=53 loops=1)
                                        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=9455.721..9455.729 rows=53 loops=1)
                                            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=9455.720..9455.720 rows=53 loops=1)
                                                -> Window aggregate: rank() OVER (ORDER BY avg(CO.co_measurement) desc )   (actual time=9455.676..9455.700 rows=53 loops=1)
                                                    -> Sort: avg_co DESC  (actual time=9455.671..9455.676 rows=53 loops=1)
                                                        -> Table scan on <temporary>  (cost=2.50..2.50 rows=0) (actual time=9455.632..9455.641 rows=53 loops=1)
                                                            -> Temporary table  (cost=0.00..0.00 rows=0) (actual time=9455.629..9455.629 rows=53 loops=1)
                                                                -> Window aggregate: rank() OVER (ORDER BY avg(AQD.aqi) desc )   (actual time=9455.519..9455.601 rows=53 loops=1)
                                                                    -> Sort: avg_aqi DESC  (actual time=9455.496..9455.503 rows=53 loops=1)
                                                                        -> Table scan on <temporary>  (actual time=9454.381..9454.402 rows=53 loops=1)
                                                                            -> Aggregate using temporary table  (actual time=9454.374..9454.374 rows=53 loops=1)
                                                                                -> Nested loop left join  (cost=11737788.23 rows=27331559) (actual time=0.151..3548.732 rows=1320030 loops=1)
                                                                                    -> Nested loop left join  (cost=3550471.30 rows=21810584) (actual time=0.138..1092.428 rows=1223088 loops=1)
                                                                                        -> Nested loop left join  (cost=451465.67 rows=3639468) (actual time=0.123..182.016 rows=203853 loops=1)
                                                                                            -> Nested loop left join  (cost=29214.37 rows=230243) (actual time=0.108..43.146 rows=19172 loops=1)
                                                                                                -> Nested loop left join  (cost=2564.37 rows=14303) (actual time=0.098..14.923 rows=6465 loops=1)
                                                                                                    -> Table scan on L  (cost=325.70 rows=3222) (actual time=0.075..2.166 rows=3222 loops=1)
                                                                                                    -> Index lookup on AQD using PRIMARY (county_code=L.county_code)  (cost=0.25 rows=4) (actual time=0.003..0.004 rows=1 loops=3222)
                                                                                                -> Index lookup on CO using PRIMARY (county_code=L.county_code)  (cost=0.25 rows=16) (actual time=0.003..0.004 rows=2 loops=6465)
                                                                                            -> Index lookup on N using PRIMARY (county_code=L.county_code)  (cost=0.25 rows=16) (actual time=0.004..0.006 rows=10 loops=19172)
                                                                                        -> Index lookup on D using PRIMARY (county_code=L.county_code)  (cost=0.25 rows=6) (actual time=0.003..0.004 rows=6 loops=203853)
                                                                                    -> Covering index lookup on U using idx_user_profile_county_code (county_code=L.county_code)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0 loops=12230
88)
 |
```

CREATE INDEX idx_co_measurement ON COData(co_measurement);



CREATE INDEX idx_no_measurement ON NO2Data(no2_measurement);



After testing different indexing strategies, a composite index was added on UserProfile(county_code, user_id), as well as individual indexes on COData(co_measurement) and NO2Data(no2_measurement). Although this approach was intended to optimize joins and the counting of users, it ultimately increased the query cost. This outcome was likely due to the added I/O and maintenance costs introduced by the new indexes, especially since user_id is often used solely for counts rather than selective filtering. Consequently, we decided to revert to the original indexing strategy, which provides a more balanced performance without the added overhead. Given this, we decided to retain the original design, which offers a more balanced performance without the added complexity.

3. Advanced Queries (3)

Identify counties where air quality (as measured by the AQI) worsened during the summer months (2024-06 to 2024-08) compared to the first quarter of the year (2024-01 to 2024-04).

```
SELECT L.state, L.county_name, AVG(AQD.aqi) AS avg_aqi, MAX(AQD_Q1.avg_aqi_Q1) AS avg_aqi_Q1
FROM AirQualityData AQD
JOIN Location L ON AQD.county_code = L.county_code
JOIN (
    SELECT county_code, AVG(aqi) AS avg_aqi_Q1
    FROM AirQualityData
    WHERE timestamp BETWEEN '2024-01' AND '2024-04'
    GROUP BY county_code
) AS AQD_Q1 ON AQD.county_code = AQD_Q1.county_code
WHERE AQD.timestamp BETWEEN '2024-06' AND '2024-08'
GROUP BY L.state, L.county_name
HAVING avg_aqi > MAX(AQD_Q1.avg_aqi_Q1)
ORDER BY L.state, L.county_name
LIMIT 15;
```

```
+-------+-------------+------------+------------+
| state | county_name | avg_aqi    | avg_aqi_Q1 |
+-------+-------------+------------+------------+
| AR    | Arkansas    | 43.9000000 | 40.4045000 |
| AR    | Ashley      | 43.7000000 | 33.7655000 |
| AR    | Crittenden  | 59.4330000 | 38.3617500 |
| AR    | Garland     | 49.9000000 | 42.4867500 |
| AR    | Jackson     | 43.1110000 | 32.5167500 |
| AR    | Newton      | 40.8670000 | 39.3980000 |
| AR    | Polk        | 42.7000000 | 41.9045000 |
| AR    | Union       | 45.3000000 | 42.0170000 |
| AR    | Washington  | 45.5670000 | 41.3535000 |
| AZ    | Apache      | 18.8000000 |  9.5377500 |
| AZ    | Gila        | 68.5670000 | 48.2180000 |
| AZ    | La Paz      | 55.7670000 | 45.1050000 |
| AZ    | Maricopa    | 75.4100000 | 61.9505000 |
| AZ    | Mohave      | 28.5000000 | 12.1230000 |
| AZ    | Navajo      | 16.7330000 |  9.7170000 |
+-------+-------------+------------+------------+
15 rows in set (0.04 sec)
```

- **Indexing Analysis**

Before:

```
| -> Limit: 15 row(s)  (actual time=10.296..10.298 rows=15 loops=1)
    -> Sort: L.state, L.county_name  (actual time=10.295..10.296 rows=15 loops=1)
        -> Filter: (avg_aqi > max(AQD_Q1.avg_aqi_Q1))  (actual time=9.950..10.140 rows=246 loops=1)
            -> Table scan on <temporary>  (actual time=9.940..10.027 rows=362 loops=1)
                -> Aggregate using temporary table  (actual time=9.938..9.938 rows=362 loops=1)
                    -> Nested loop inner join  (cost=926.06 rows=229) (actual time=4.138..8.891 rows=474 loops=1)
                        -> Nested loop inner join  (cost=845.79 rows=229) (actual time=4.122..8.314 rows=474 loops=1)
                            -> Table scan on AQD_Q1  (cost=514.38..522.68 rows=465) (actual time=4.012..4.157 rows=943 loops=1)
                                -> Materialize  (cost=514.36..514.36 rows=465) (actual time=4.008..4.008 rows=943 loops=1)
                                    -> Group aggregate: avg(AirQualityData.aqi)  (cost=467.86 rows=465) (actual time=0.073..3.611 rows=943 loops=1)
                                        -> Filter: (AirQualityData.`timestamp` between '2024-01' and '2024-04')  (cost=421.35 rows=465) (actual time=0.062..2.989 rows=3223 loops=1
)
                                            -> Index scan on AirQualityData using PRIMARY  (cost=421.35 rows=4186) (actual time=0.057..1.397 rows=4186 loops=1)
                            -> Filter: (AQD.`timestamp` between '2024-06' and '2024-08')  (cost=0.25 rows=0.5) (actual time=0.004..0.004 rows=1 loops=943)
                                -> Index lookup on AQD using PRIMARY (county_code=AQD_Q1.county_code)  (cost=0.25 rows=4) (actual time=0.002..0.003 rows=4 loops=943)
                        -> Single-row index lookup on L using PRIMARY (county_code=AQD_Q1.county_code)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=474)
 |
```

CREATE INDEX idx_aqd_county_timestamp ON AirQualityData(county_code, timestamp);

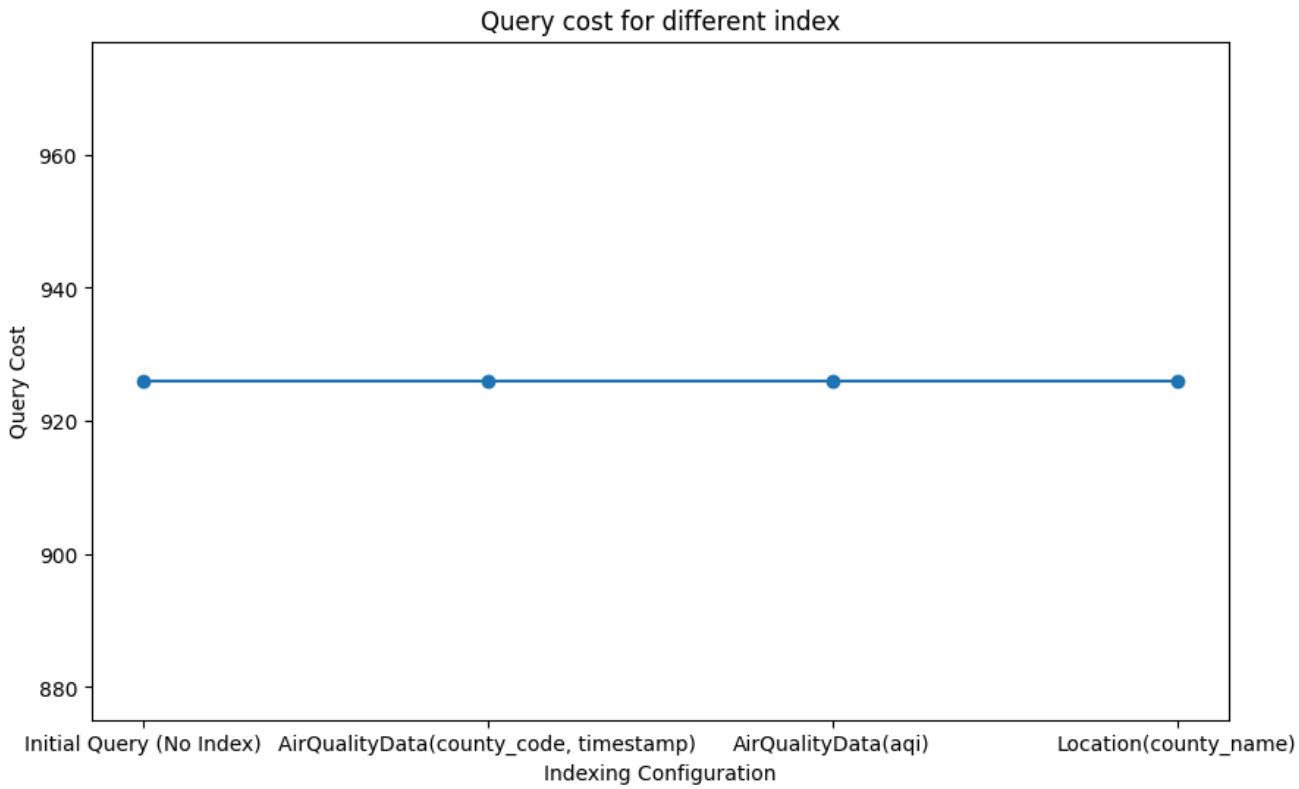
```
| -> Limit: 15 row(s)  (actual time=9.639..9.641 rows=15 loops=1)
    -> Sort: L.state, L.county_name  (actual time=9.638..9.639 rows=15 loops=1)
        -> Filter: (avg_aqi > max(AQD_Q1.avg_aqi_Q1))  (actual time=9.358..9.510 rows=246 loops=1)
            -> Table scan on <temporary>  (actual time=9.347..9.401 rows=362 loops=1)
                -> Aggregate using temporary table  (actual time=9.344..9.344 rows=362 loops=1)
                    -> Nested loop inner join  (cost=926.06 rows=229) (actual time=4.314..8.562 rows=474 loops=1)
                        -> Nested loop inner join  (cost=845.79 rows=229) (actual time=4.305..8.082 rows=474 loops=1)
                            -> Table scan on AQD_Q1  (cost=514.38..522.68 rows=465) (actual time=4.166..4.292 rows=943 loops=1)
                                -> Materialize  (cost=514.36..514.36 rows=465) (actual time=4.162..4.162 rows=943 loops=1)
                                    -> Group aggregate: avg(AirQualityData.aqi)  (cost=467.86 rows=465) (actual time=0.098..3.790 rows=943 loops=1)
                                        -> Filter: (AirQualityData.`timestamp` between '2024-01' and '2024-04')  (cost=421.35 rows=465) (actual time=0.085..3.106 rows=3223 loops=1
)
                                            -> Index scan on AirQualityData using PRIMARY  (cost=421.35 rows=4186) (actual time=0.078..1.522 rows=4186 loops=1)
                            -> Filter: (AQD.`timestamp` between '2024-06' and '2024-08')  (cost=0.25 rows=0.5) (actual time=0.004..0.004 rows=1 loops=943)
                                -> Index lookup on AQD using PRIMARY (county_code=AQD_Q1.county_code)  (cost=0.25 rows=4) (actual time=0.002..0.003 rows=4 loops=943)
                        -> Single-row index lookup on L using PRIMARY (county_code=AQD_Q1.county_code)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=474)
 |
```

CREATE INDEX idx_aqi ON AirQualityData(aqi);

```
| -> Limit: 15 row(s)  (actual time=12.463..12.465 rows=15 loops=1)
    -> Sort: L.state, L.county_name  (actual time=12.462..12.463 rows=15 loops=1)
        -> Filter: (avg_aqi > max(AQD_Q1.avg_aqi_Q1))  (actual time=9.891..10.098 rows=246 loops=1)
            -> Table scan on <temporary>  (actual time=9.883..9.967 rows=362 loops=1)
                -> Aggregate using temporary table  (actual time=9.880..9.880 rows=362 loops=1)
                    -> Nested loop inner join  (cost=926.06 rows=229) (actual time=4.420..8.958 rows=474 loops=1)
                        -> Nested loop inner join  (cost=845.79 rows=229) (actual time=4.412..8.440 rows=474 loops=1)
                            -> Table scan on AQD_Q1  (cost=514.38..522.68 rows=465) (actual time=4.311..4.456 rows=943 loops=1)
                                -> Materialize  (cost=514.36..514.36 rows=465) (actual time=4.308..4.308 rows=943 loops=1)
                                    -> Group aggregate: avg(AirQualityData.aqi)  (cost=467.86 rows=465) (actual time=0.878..3.991 rows=943 loops=1)
                                        -> Filter: (AirQualityData.`timestamp` between '2024-01' and '2024-04')  (cost=421.35 rows=465) (actual time=0.863..3.355 rows=3223 loops=1
)
                                            -> Index scan on AirQualityData using PRIMARY  (cost=421.35 rows=4186) (actual time=0.076..1.198 rows=4186 loops=1)
                            -> Filter: (AQD.`timestamp` between '2024-06' and '2024-08')  (cost=0.25 rows=0.5) (actual time=0.004..0.004 rows=1 loops=943)
                                -> Index lookup on AQD using PRIMARY (county_code=AQD_Q1.county_code)  (cost=0.25 rows=4) (actual time=0.002..0.003 rows=4 loops=943)
                        -> Single-row index lookup on L using PRIMARY (county_code=AQD_Q1.county_code)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=474)
```

CREATE INDEX idx_lc_countyname ON Location(county_name);

```
| -> Limit: 15 row(s)  (actual time=9.354..9.357 rows=15 loops=1)
    -> Sort: L.state, L.county_name  (actual time=9.353..9.354 rows=15 loops=1)
        -> Filter: (avg_aqi > max(AQD_Q1.avg_aqi_Q1))  (actual time=8.932..9.108 rows=246 loops=1)
            -> Table scan on <temporary>  (actual time=8.921..8.989 rows=362 loops=1)
                -> Aggregate using temporary table  (actual time=8.919..8.919 rows=362 loops=1)
                    -> Nested loop inner join  (cost=926.06 rows=229) (actual time=3.706..8.100 rows=474 loops=1)
                        -> Nested loop inner join  (cost=845.79 rows=229) (actual time=3.695..7.591 rows=474 loops=1)
                            -> Table scan on AQD_Q1  (cost=514.38..522.68 rows=465) (actual time=3.572..3.714 rows=943 loops=1)
                                -> Materialize  (cost=514.36..514.36 rows=465) (actual time=3.569..3.569 rows=943 loops=1)
                                    -> Group aggregate: avg(AirQualityData.aqi)  (cost=467.86 rows=465) (actual time=0.049..3.256 rows=943 loops=1)
                                        -> Filter: (AirQualityData.`timestamp` between '2024-01' and '2024-04')  (cost=421.35 rows=465) (actual time=0.041..2.610 rows=3223 loops=1
)
                                            -> Index scan on AirQualityData using PRIMARY  (cost=421.35 rows=4186) (actual time=0.037..1.203 rows=4186 loops=1)
                            -> Filter: (AQD.`timestamp` between '2024-06' and '2024-08')  (cost=0.25 rows=0.5) (actual time=0.004..0.004 rows=1 loops=943)
                                -> Index lookup on AQD using PRIMARY (county_code=AQD_Q1.county_code)  (cost=0.25 rows=4) (actual time=0.002..0.003 rows=4 loops=943)
                        -> Single-row index lookup on L using PRIMARY (county_code=AQD_Q1.county_code)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=474)
|
```



The optimized indexing strategy effectively enhances the query's efficiency in identifying counties with worsening air quality over specific time periods by targeting the key areas of filtering and joining. The composite index on AirQualityData(county_code, timestamp) allows the database to retrieve data for specific date ranges quickly without performing a full table scan, thus reducing I/O costs and improving query speed. Additionally, the index on county_name in the Location table supports faster access to county names during joins, further improving performance.

An index on aqi (idx_aqi) was initially tested but was ultimately excluded as it caused performance degradation. This was likely due to the overhead of maintaining the index on a column that is frequently aggregated rather than filtered. The additional I/O required to update and reference the aqi index negated any potential gains, as the query spends more time on aggregating aqi than directly accessing individual values. By excluding idx_aqi, the final indexing strategy effectively balances performance, making it scalable and well-suited for complex environmental analysis across large datasets.

4. Advanced Queries (4)

Search the average drought level for each user's county from 2023-12 to 2024-05

```
SELECT u.username, l.county_name, AVG(d.drought_level) AS average_drought_level
FROM UserProfile u
JOIN
    Location l ON u.county_code = l.county_code
JOIN
    DroughtData d ON u.county_code = d.county_code
WHERE
    d.timestamp BETWEEN '2023-12' AND '2024-05'
GROUP BY
    u.username, l.county_name
ORDER BY
    u.username
LIMIT 15;
```

username	county_name	average_drought_level
aagzzdxp	East Baton Rouge	12.5615000
aaixzwmg	Alamosa	32.4050000
aakqnirr	Le Flore County	0.9936667
aalpfvae	Warren County	42.3340000
abbjobgh	Queen Anne's County	0.0000000
abmbqhpo	Freestone County	3.5110000
abwewtnb	Golden Valley County	5.6885000
adbvklgq	Niobrara County	2.4715000
adkjcnpj	Seneca County	2.8721667
aduaqexx	Aguadilla Municipio	12.7423333
aevcyoia	Onslow County	0.8876667
aflapeic	Poinsett County	5.2045000
aflcjnoj	Bandera County	57.7248333
afvbcfky	Oglethorpe County	4.9920000
afwahlxt	Rusk County	7.5388333

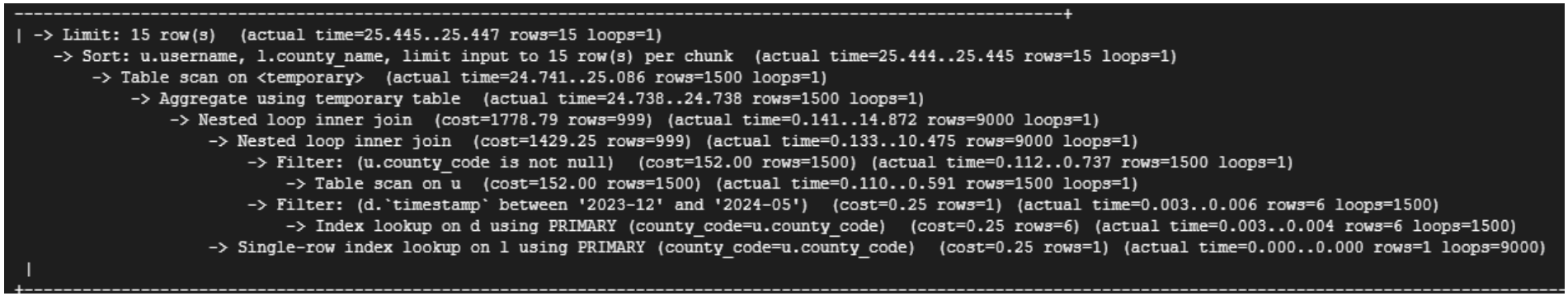
15 rows in set (0.02 sec)

- **Indexing Analysis**

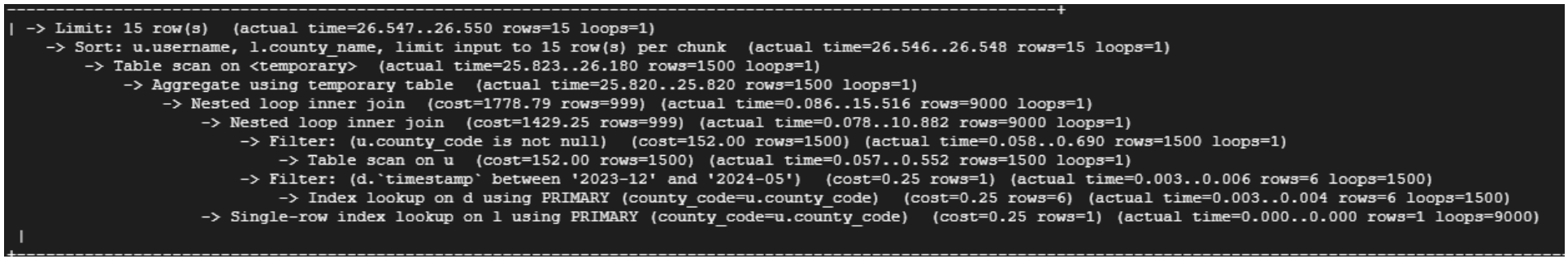
Before adding index:

```
-> Limit: 15 row(s)  (actual time=27.398..27.400 rows=15 loops=1)
    -> Sort: u.username, l.county_name, limit input to 15 row(s) per chunk  (actual time=27.397..27.398 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=25.765..26.130 rows=1500 loops=1)
            -> Aggregate using temporary table  (actual time=25.762..25.762 rows=1500 loops=1)
                -> Nested loop inner join  (cost=1778.79 rows=999) (actual time=0.114..15.432 rows=9000 loops=1)
                    -> Nested loop inner join  (cost=1429.25 rows=999) (actual time=0.106..10.831 rows=9000 loops=1)
                        -> Filter: (u.county_code is not null)  (cost=152.00 rows=1500) (actual time=0.077..0.741 rows=1500 loops=1)
                            -> Table scan on u  (cost=152.00 rows=1500) (actual time=0.073..0.609 rows=1500 loops=1)
                        -> Filter: (d.`timestamp` between '2023-12' and '2024-05')  (cost=0.25 rows=1) (actual time=0.003..0.006 rows=6 loops=1500)
                            -> Index lookup on d using PRIMARY (county_code=u.county_code)  (cost=0.25 rows=6) (actual time=0.003..0.004 rows=6 loops=1500)
                    -> Single-row index lookup on l using PRIMARY (county_code=u.county_code)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=9000)
```

Add first index:
CREATE INDEX idx_user_profile_county_code ON UserProfile(county_code);

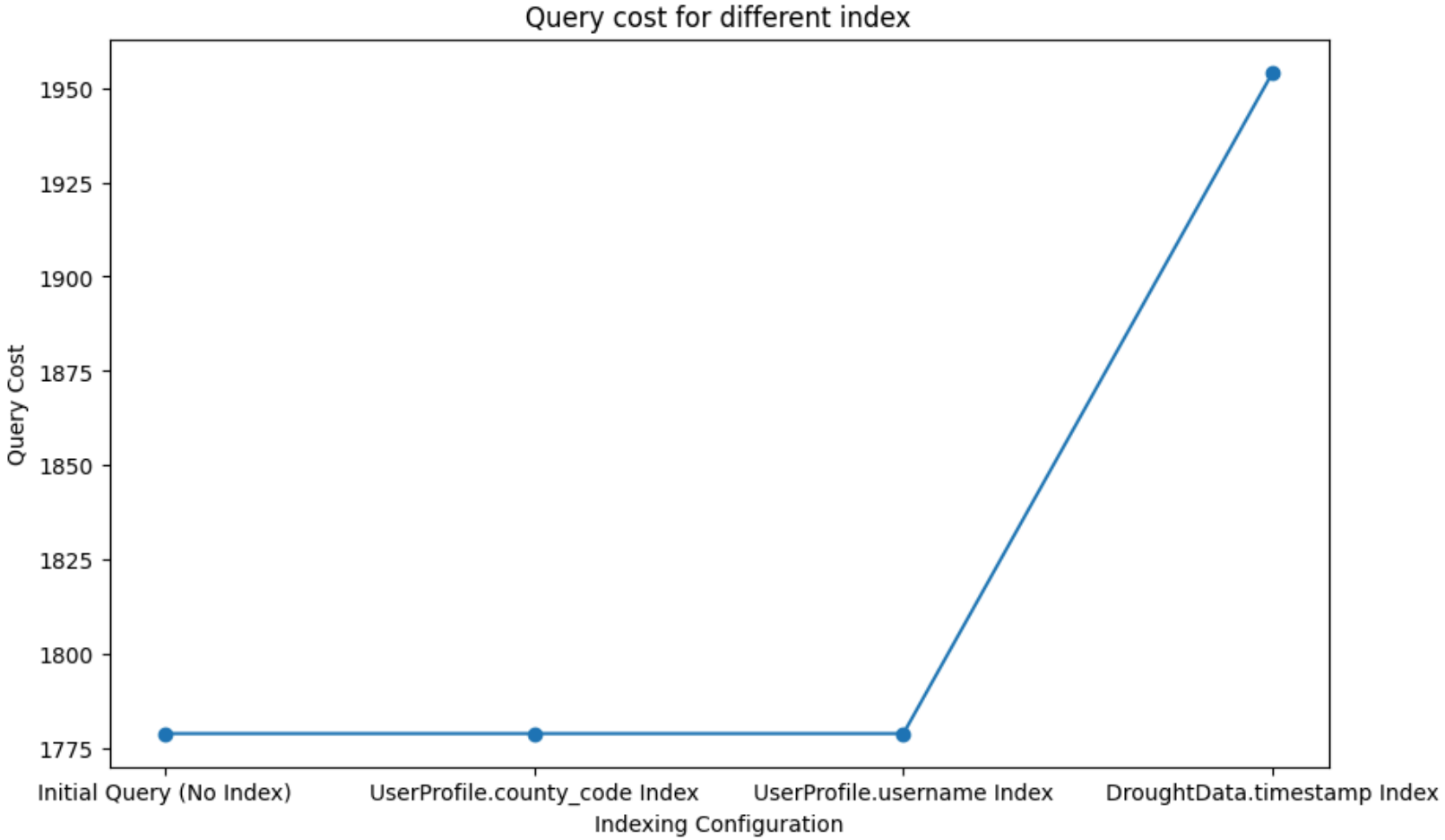


Add Second index:
CREATE INDEX idx_user_profile_username ON UserProfile(username);



Add third index:
CREATE INDEX idx_drought_data_timestamp ON DroughtData(timestamp);

```
-> Limit: 15 row(s)  (actual time=24.675..24.677 rows=15 loops=1)
    -> Sort: u.username, l.county_name, limit input to 15 row(s) per chunk  (actual time=24.674..24.675 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=23.982..24.320 rows=1500 loops=1)
            -> Aggregate using temporary table  (actual time=23.980..23.980 rows=1500 loops=1)
                -> Nested loop inner join  (cost=1954.25 rows=4494) (actual time=0.080..13.137 rows=9000 loops=1)
                    -> Nested loop inner join  (cost=677.00 rows=1500) (actual time=0.065..3.132 rows=1500 loops=1)
                        -> Filter: (u.county_code is not null)  (cost=152.00 rows=1500) (actual time=0.053..0.683 rows=1500 loops=1)
                            -> Table scan on u  (cost=152.00 rows=1500) (actual time=0.052..0.534 rows=1500 loops=1)
                        -> Single-row index lookup on l using PRIMARY (county_code=u.county_code)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1500)
                    -> Filter: (d.`timestamp` between '2023-12' and '2024-05')  (cost=0.25 rows=3) (actual time=0.003..0.006 rows=6 loops=1500)
                        -> Index lookup on d using PRIMARY (county_code=u.county_code)  (cost=0.25 rows=6) (actual time=0.003..0.004 rows=6 loops=1500)
```



After we add indexes of UserProfile.county_code and UserProfile.username, we are hardly able to find any change of cost.

The reasons might be that the MySQL optimizer may consider a full table scan to be less costly than using an index, especially if the JOIN tables are similar in size. Also, I guess that another reason might be that the data size is not big which causes the advantages of indexes to be not obvious.

However, after we added the index of DroughtData.timestamp, we found an obvious increase in cost. In some cases, indexing can increase the cost of a query, especially when aggregation (such as AVG) and sorting operations are involved, and the database may require additional processing of the query data. Due to the use of timestamp indexes, the database may create more temporary tables internally or perform additional sorting, resulting in increased query costs.

In conclusion, I will choose the indexes of UserProfile.county_code or UserProfile.username, since these indexes will not increase the cost.