Part 1:

Database Implementation

We implemented our database on a MySQL instance on GCP since it would be easier to access the database as compared to everyone having four separate local instances.

```
tylercraigxc@cloudshell:~ (cs411-group-109-database)$ gcloud sql connect group109 --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 1613
Server version: 8.0.37-google (Google)

Copyright (c) 2000, 2025, 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> use oncelerwatch
```

```
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_oncelerwatch  |
+-------------------------+
| country_data            |
| favorites               |
| locations               |
| search_history          |
| subnational_data        |
| users                   |
+-------------------------+
6 rows in set (0.00 sec)
```

Database Design (DDL)

```
CREATE TABLE Users (
        userId INT AUTO_INCREMENT PRIMARY KEY,
        email VARCHAR(255) NOT NULL UNIQUE,
        username VARCHAR(255) NOT NULL UNIQUE,
       password VARCHAR(255) NOT NULL
);
```

```
CREATE TABLE country_data (
    location_id INT PRIMARY KEY,
    views INT,
    country VARCHAR(255),
    umd_tree_cover_extent_2000__ha BIGINT,
    gfw_aboveground_carbon_stocks_2000__Mg_C BIGINT,
    avg_gfw_aboveground_carbon_stocks_2000__Mg_C_ha FLOAT,
    gfw_forest_carbon_gross_emissions__Mg_CO2e_yr FLOAT,
    gfw_forest_carbon_gross_removals__Mg_CO2_yr FLOAT,
    gfw_forest_carbon_net_flux__Mg_CO2e_yr FLOAT,
    gfw_forest_carbon_gross_emissions_2023__Mg_CO2e FLOAT,
    area_ha BIGINT,
    tc_loss_ha_2023 BIGINT,
    primary_loss_ha_2023 BIGINT,
    CONSTRAINT fk_location_id FOREIGN KEY (location_id) REFERENCES
locations(location_id) ON DELETE CASCADE
);

CREATE TABLE subnational_data (
    location_id INT PRIMARY KEY,
    views INT,
    country VARCHAR(255),
    subnational1 VARCHAR(255),
    umd_tree_cover_extent_2000__ha BIGINT,
    gfw_aboveground_carbon_stocks_2000__Mg_C BIGINT,
    avg_gfw_aboveground_carbon_stocks_2000__Mg_C_ha FLOAT,
    gfw_forest_carbon_gross_emissions__Mg_CO2e_yr FLOAT,
    gfw_forest_carbon_gross_removals__Mg_CO2_yr FLOAT,
    gfw_forest_carbon_net_flux__Mg_CO2e_yr FLOAT,
    gfw_forest_carbon_gross_emissions_2023__Mg_CO2e FLOAT,
    area_ha BIGINT,
    tc_loss_ha_2023 BIGINT,
    primary_loss_ha_2023 BIGINT,
    CONSTRAINT fk_location_id FOREIGN KEY (location_id) REFERENCES
locations(location_id) ON DELETE CASCADE
);

CREATE TABLE favorites (
    userId INT,
    location_id INT,
    PRIMARY KEY (userId, location_id),
    CONSTRAINT fk_userId FOREIGN KEY (userId) REFERENCES Users(userId) ON DELETE
CASCADE,
```

```
    CONSTRAINT fk_location_id FOREIGN KEY (location_id) REFERENCES
locations(location_id) ON DELETE CASCADE
);
CREATE TABLE search_history (
    userId INT,
    location_id INT,
    searchOrderNum INT,
    PRIMARY KEY (userId, location_id, searchOrderNum),
    CONSTRAINT fk_search_userId FOREIGN KEY (userId) REFERENCES Users(userId) ON
DELETE CASCADE,
    CONSTRAINT fk_search_location_id FOREIGN KEY (location_id) REFERENCES
locations(location_id) ON DELETE CASCADE
);

CREATE TABLE locations (
     location_id INT PRIMARY KEY
);
```

Post Insertion:

```
Database changed
mysql> SELECT COUNT(*) FROM locations
    -> ;
+----------+
| COUNT(*) |
+----------+
|     3777 |
+----------+
```

```
mysql> SELECT COUNT(*) FROM subnational_data;
+----------+
| COUNT(*) |
+----------+
|     3541 |
+----------+
```

```
mysql> SELECT COUNT(*) FROM users;
+----------+
| COUNT(*) |
+----------+
|     1964 |
+----------+
```

Queries

Query#1 See if the country is a net emitter or a net absorber of carbon

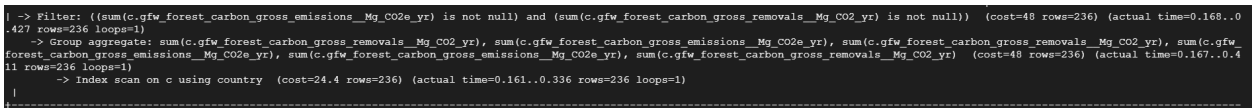
```
mysql> SELECT c.country,
    ->        SUM(c.gfw_forest_carbon_gross_emissions__Mg_CO2e_yr) AS total_emissions,
    ->        SUM(c.gfw_forest_carbon_gross_removals__Mg_CO2_yr) AS total_removals,
    ->        CASE
    ->            WHEN SUM(c.gfw_forest_carbon_gross_emissions__Mg_CO2e_yr) > SUM(c.gfw_forest_carbon_gross_removals__Mg_CO2_yr)
    ->            THEN 'Net Carbon Emitter'
    ->            ELSE 'Net Carbon Absorber'
    ->        END AS status
    -> FROM country_data c
    -> GROUP BY c.country
    -> HAVING SUM(c.gfw_forest_carbon_gross_emissions__Mg_CO2e_yr) IS NOT NULL
    ->    AND SUM(c.gfw_forest_carbon_gross_removals__Mg_CO2_yr) IS NOT NULL
    -> LIMIT 15;
+----------------------+-----------------+----------------+--------------------+
| country              | total_emissions | total_removals | status             |
+----------------------+-----------------+----------------+--------------------+
| Akrotiri and Dhekelia |             636 |           1628 | Net Carbon Absorber |
| Albania              |          723550 |        5114214 | Net Carbon Absorber |
| Algeria              |         2779935 |        4904868 | Net Carbon Absorber |
| Andorra              |            2764 |          93635 | Net Carbon Absorber |
| Angola               |        59427204 |      170958560 | Net Carbon Absorber |
| Anguilla             |             440 |           8332 | Net Carbon Absorber |
| Antigua and Barbuda  |           12744 |          94090 | Net Carbon Absorber |
| Argentina            |        71979632 |      163615504 | Net Carbon Absorber |
| Armenia              |           30703 |         901660 | Net Carbon Absorber |
| Aruba                |              16 |            533 | Net Carbon Absorber |
| Australia            |        97242264 |      207229488 | Net Carbon Absorber |
| Austria              |         9217507 |       29561502 | Net Carbon Absorber |
| Azerbaijan           |           99246 |        4355668 | Net Carbon Absorber |
| Bahamas              |          424457 |        2847847 | Net Carbon Absorber |
| Bahrain              |               0 |              0 | Net Carbon Absorber |
+----------------------+-----------------+----------------+--------------------+
15 rows in set (0.00 sec)
```

```
SELECT c.country,
    SUM(c.gfw_forest_carbon_gross_emissions__Mg_CO2e_yr) AS total_emissions,
    SUM(c.gfw_forest_carbon_gross_removals__Mg_CO2_yr) AS total_removals,
    CASE
      WHEN SUM(c.gfw_forest_carbon_gross_emissions__Mg_CO2e_yr) >
SUM(c.gfw_forest_carbon_gross_removals__Mg_CO2_yr)
      THEN 'Net Carbon Emitter'
      ELSE 'Net Carbon Absorber'
    END AS status
FROM country_data c
GROUP BY c.country
HAVING SUM(c.gfw_forest_carbon_gross_emissions__Mg_CO2e_yr) IS NOT NULL
  AND SUM(c.gfw_forest_carbon_gross_removals__Mg_CO2_yr) IS NOT NULL;
```

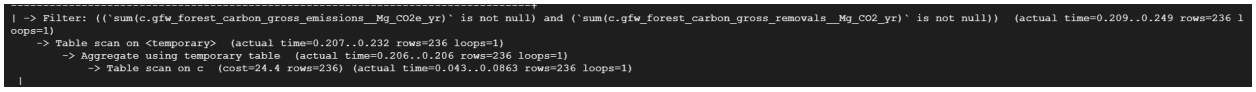
```
| -> Filter: ((`sum(c.gfw_forest_carbon_gross_emissions__Mg_CO2e_yr)` is not null) and (`sum(c.gfw_forest_carbon_gross_removals__Mg_CO2_yr)` is not null))  (actual time=0.214..0.276 rows=236 l
oops=1)
    -> Table scan on <temporary>  (actual time=0.211..0.257 rows=236 loops=1)
        -> Aggregate using temporary table  (actual time=0.211..0.211 rows=236 loops=1)
            -> Table scan on c  (cost=24.4 rows=236) (actual time=0.0472..0.0906 rows=236 loops=1)
```

```
CREATE INDEX country ON country_data(country);
CREATE INDEX emissions ON
country_data(gfw_forest_carbon_gross_emissions__Mg_CO2e_yr);
CREATE INDEX removals ON
country_data(gfw_forest_carbon_gross_removals__Mg_CO2_yr);
```

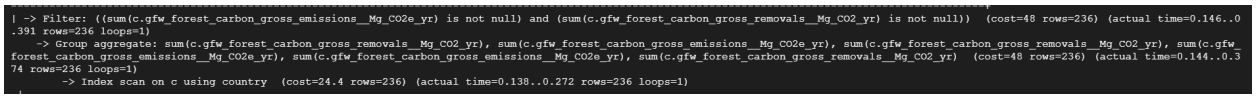
INDEX country:



INDEX emissions, removals:



INDEX country, emissions, removals:



ANALYSIS:
As seen by the cost from each of these indices, indexing country, emissions, and removals are the best as seen by the faster runtime, as there is not much cost computation to go off of. However, there is an optimization in query 4 that we have that removes many more tokens than this query optimization does, and because both queries will be called around the same number of times, we have decided not to index on country and just index on emissions and removals. We will still index emissions and removals in case we need to use the indexing in the future, as the runtime of emissions and removals vs no index is about the same.

Query#2 Get the rank of the country based on tree loss: primary loss ratio

```
mysql> SELECT c.country,
    ->        SUM(c.tc_loss_ha_2023) AS net_loss,
    ->        RANK() OVER (ORDER BY SUM(c.tc_loss_ha_2023) DESC) AS rank_position
    -> FROM country_data c
    -> GROUP BY c.country
    -> LIMIT 15;
+---------------------------------+----------+---------------+
| country                         | net_loss | rank_position |
+---------------------------------+----------+---------------+
| Canada                          |  8570168 |             1 |
| Russia                          |  3354720 |             2 |
| Brazil                          |  2806058 |             3 |
| Indonesia                       |  1395285 |             4 |
| United States                   |  1382421 |             5 |
| Democratic Republic of the Congo |  1324890 |             6 |
| Bolivia                         |   696363 |             7 |
| China                           |   602595 |             8 |
| Laos                            |   445467 |             9 |
| Sweden                          |   313926 |            10 |
| Malaysia                        |   308585 |            11 |
| Myanmar                         |   307184 |            12 |
| Madagascar                      |   303190 |            13 |
| Angola                          |   268336 |            14 |
| Mozambique                      |   262176 |            15 |
+---------------------------------+----------+---------------+
15 rows in set (0.00 sec)
```

```
SELECT c.country,
    SUM(c.tc_loss_ha_2023) AS net_loss,
    RANK() OVER (ORDER BY SUM(c.tc_loss_ha_2023) DESC) AS rank_position
FROM country_data c
GROUP BY c.country
HAVING SUM(c.tc_loss_ha_2023) > 0 AND SUM(c.primary_loss_ha_2023) IS NOT NULL;
```

```
| -> Window aggregate: rank() OVER (ORDER BY net_loss desc )   (actual time=0.366..0.424 rows=97 loops=1)
    -> Filter: ((`sum(c.tc_loss_ha_2023)` > 0) and (`sum(c.primary_loss_ha_2023)` is not null))  (actual time=0.363..0.399 rows=97 loops=1)
        -> Sort: net_loss DESC  (actual time=0.361..0.374 rows=236 loops=1)
            -> Table scan on <temporary>  (actual time=0.27..0.294 rows=236 loops=1)
                -> Aggregate using temporary table  (actual time=0.269..0.269 rows=236 loops=1)
                    -> Table scan on c  (cost=24.4 rows=236) (actual time=0.0571..0.103 rows=236 loops=1)
|
```

```
CREATE INDEX country ON country_data(country);
CREATE INDEX tc_loss ON country_data(tc_loss_ha_2023);
CREATE INDEX primary_loss ON country_data(primary_loss_ha_2023);
```

INDEX country:

```
-> Window aggregate: rank() OVER (ORDER BY net_loss desc )   (actual time=0.367..0.424 rows=97 loops=1)
  -> Filter: ((`sum(c.tc_loss_ha_2023)` > 0) and (`sum(c.primary_loss_ha_2023)` is not null))  (actual time=0.364..0.399 rows=97 loops=1)
      -> Sort: net_loss DESC  (actual time=0.361..0.375 rows=236 loops=1)
          -> Table scan on <temporary>  (actual time=0.253..0.295 rows=236 loops=1)
              -> Aggregate using temporary table  (actual time=0.252..0.252 rows=236 loops=1)
                  -> Table scan on c  (cost=24.4 rows=236) (actual time=0.0481..0.0925 rows=236 loops=1)
|
```

INDEX tc_loss, primary_loss:

```
| -> Window aggregate: rank() OVER (ORDER BY net_loss desc )   (actual time=0.332..0.389 rows=97 loops=1)
    -> Filter: ((`sum(c.tc_loss_ha_2023)` > 0) and (`sum(c.primary_loss_ha_2023)` is not null))  (actual time=0.33..0.366 rows=97 loops=1)
        -> Sort: net_loss DESC  (actual time=0.327..0.341 rows=236 loops=1)
            -> Table scan on <temporary>  (actual time=0.235..0.261 rows=236 loops=1)
                -> Aggregate using temporary table  (actual time=0.234..0.234 rows=236 loops=1)
                    -> Table scan on c  (cost=24.4 rows=236) (actual time=0.0453..0.0887 rows=236 loops=1)
 |
```

INDEX tc_loss, primary_loss, country:

```
| -> Window aggregate: rank() OVER (ORDER BY net_loss desc )   (actual time=0.335..0.391 rows=97 loops=1)
    -> Filter: ((`sum(c.tc_loss_ha_2023)` > 0) and (`sum(c.primary_loss_ha_2023)` is not null))  (actual time=0.332..0.367 rows=97 loops=1)
        -> Sort: net_loss DESC  (actual time=0.329..0.343 rows=236 loops=1)
            -> Table scan on <temporary>  (actual time=0.239..0.265 rows=236 loops=1)
                -> Aggregate using temporary table  (actual time=0.238..0.238 rows=236 loops=1)
                    -> Table scan on c  (cost=24.4 rows=236) (actual time=0.047..0.0914 rows=236 loops=1)
 |
```

ANALYSIS:

As seen by the cost from each of these indices, there is no apparent difference in cost in each query. Because we see no difference, but there may be applications where indexing could be useful, we will move forward with indexing tc_loss and primary_loss. As an additional note, the runtime is also very similar for each index technique, so I believe there is no significant difference. We will not index the country column as it is more useful for query 4 not to index it.

Query#3 Get subnations whose above-ground carbon stocks are below the national average

```
mysql> SELECT s.subnational1,
    ->        SUM(s.gfw_aboveground_carbon_stocks_2000__Mg_C) AS total_carbon_stocks
    -> FROM subnational_data s
    -> GROUP BY s.subnational1
    -> HAVING SUM(s.gfw_aboveground_carbon_stocks_2000__Mg_C) < (
    ->     SELECT AVG(gfw_aboveground_carbon_stocks_2000__Mg_C)
    ->     FROM country_data
    -> )
    -> LIMIT 15;
+--------------+----------------------+
| subnational1 | total_carbon_stocks  |
+--------------+----------------------+
| !Karas       |                  329 |
| Aargau       |              5139419 |
| Abia         |              6294451 |
| Abidjan      |              9841420 |
| Abkhazia     |             50441690 |
| Abra         |             29242503 |
| Abruzzo      |             22929721 |
| Absheron     |               937656 |
| Abu Dhabi    |                    0 |
| Abyan        |                    0 |
| Aceh         |            653101559 |
| Acklins      |               462943 |
| Acoua        |               120893 |
| Acquaviva    |                 3600 |
| Ad Dakhliyah |                    0 |
+--------------+----------------------+
15 rows in set (0.00 sec)
```

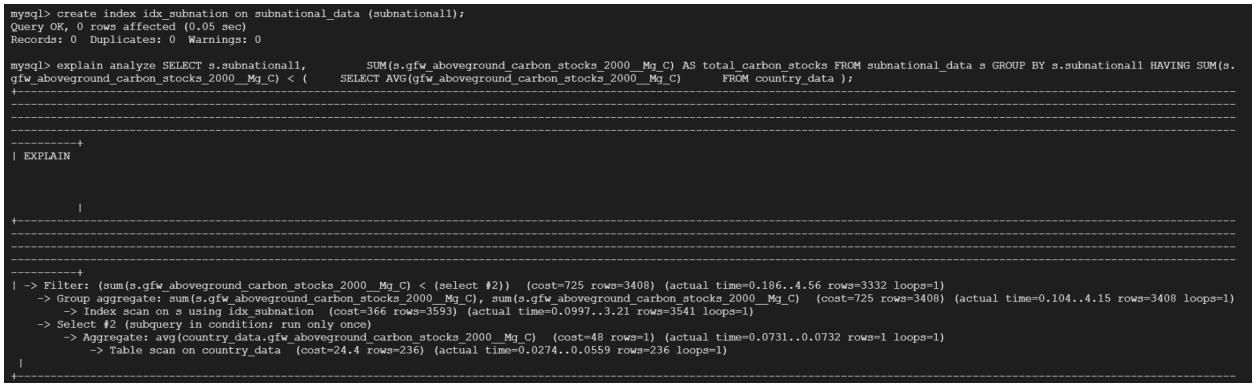
```
SELECT s.subnational1,
      SUM(s.gfw_aboveground_carbon_stocks_2000__Mg_C) AS total_carbon_stocks
FROM subnational_data s
GROUP BY s.subnational1
HAVING SUM(s.gfw_aboveground_carbon_stocks_2000__Mg_C) < (
   SELECT AVG(gfw_aboveground_carbon_stocks_2000__Mg_C)
   FROM country_data
);
```

No Indexing:

```
| -> Filter: (`sum(s.gfw_aboveground_carbon_stocks_2000__Mg_C)` < (select #2))  (actual time=2.96..3.64 rows=3332 loops=1)
    -> Table scan on <temporary>  (actual time=2.87..3.2 rows=3408 loops=1)
        -> Aggregate using temporary table  (actual time=2.87..2.87 rows=3408 loops=1)
            -> Table scan on s  (cost=366 rows=3593) (actual time=0.0462..0.88 rows=3541 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(country_data.gfw_aboveground_carbon_stocks_2000__Mg_C)  (cost=48 rows=1) (actual time=0.0693..0.0694 rows=1 loops=1)
            -> Table scan on country_data  (cost=24.4 rows=236) (actual time=0.0239..0.0513 rows=236 loops=1)
 |
```

Indexing on subnational_data.subnational1 attribute:

CREATE INDEX idx_subnational1 on subnational_data (subnational1)



```
mysql> create index idx_subnation on subnational_data (subnational1);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT s.subnational1,         SUM(s.gfw_aboveground_carbon_stocks_2000__Mg_C) AS total_carbon_stocks FROM subnational_data s GROUP BY s.subnational1 HAVING SUM(s.
gfw_aboveground_carbon_stocks_2000__Mg_C) < (     SELECT AVG(gfw_aboveground_carbon_stocks_2000__Mg_C)      FROM country_data );
+----------
| EXPLAIN


          |
+----------
| -> Filter: (sum(s.gfw_aboveground_carbon_stocks_2000__Mg_C) < (select #2))  (cost=725 rows=3408) (actual time=0.186..4.56 rows=3332 loops=1)
    -> Group aggregate: sum(s.gfw_aboveground_carbon_stocks_2000__Mg_C), sum(s.gfw_aboveground_carbon_stocks_2000__Mg_C)  (cost=725 rows=3408) (actual time=0.104..4.15 rows=3408 loops=1)
        -> Index scan on s using idx_subnation  (cost=366 rows=3593) (actual time=0.0997..3.21 rows=3541 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(country_data.gfw_aboveground_carbon_stocks_2000__Mg_C)  (cost=48 rows=1) (actual time=0.0731..0.0732 rows=1 loops=1)
            -> Table scan on country_data  (cost=24.4 rows=236) (actual time=0.0274..0.0559 rows=236 loops=1)
 |
+----------
```

Indexing on subnational_data.gfw_aboveground_carbon_stocks_2000_Mg_C:

CREATE INDEX idx_subcarbon on subnational_data
(gfw_aboveground_carbon_stocks_2000_Mg_C)

```
mysql> create idx_subcarbon on subnational_data (gfw_aboveground_carbon_stocks_2000__Mg_C);
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'idx_subcarbon on subnational_da
ta (gfw_aboveground_carbon_stocks_2000__Mg_C)' at line 1
mysql> create index idx_subcarbon on subnational_data (gfw_aboveground_carbon_stocks_2000__Mg_C);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze SELECT s.subnational1,        SUM(s.gfw_aboveground_carbon_stocks_2000__Mg_C) AS total_carbon_stocks FROM subnational_data s GROUP BY s.subnational1 HAVING SUM(s.
gfw_aboveground_carbon_stocks_2000__Mg_C) < (     SELECT AVG(gfw_aboveground_carbon_stocks_2000__Mg_C)      FROM country_data );
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                           |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------+
| -> Filter: (`sum(s.gfw_aboveground_carbon_stocks_2000__Mg_C)` < (select #2))  (actual time=2.87..3.53 rows=3332 loops=1)
    -> Table scan on <temporary>  (actual time=2.78..3.1 rows=3408 loops=1)
        -> Aggregate using temporary table  (actual time=2.78..2.78 rows=3408 loops=1)
            -> Table scan on s  (cost=366 rows=3593) (actual time=0.0383..0.833 rows=3541 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(country_data.gfw_aboveground_carbon_stocks_2000__Mg_C)  (cost=48 rows=1) (actual time=0.0771..0.0772 rows=1 loops=1)
            -> Table scan on country_data  (cost=24.4 rows=236) (actual time=0.0277..0.0552 rows=236 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Indexing on country_data.country attribute:
CREATE INDEX idx_country on country_data (country)

```
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN

                                                                                                                           |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------+
| -> Filter: (`sum(s.gfw_aboveground_carbon_stocks_2000__Mg_C)` < (select #2))  (actual time=2.8..3.46 rows=3332 loops=1)
    -> Table scan on <temporary>  (actual time=2.72..3.03 rows=3408 loops=1)
        -> Aggregate using temporary table  (actual time=2.71..2.71 rows=3408 loops=1)
            -> Table scan on s  (cost=366 rows=3593) (actual time=0.0383..0.837 rows=3541 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(country_data.gfw_aboveground_carbon_stocks_2000__Mg_C)  (cost=48 rows=1) (actual time=0.0751..0.0752 rows=1 loops=1)
            -> Table scan on country_data  (cost=24.4 rows=236) (actual time=0.0298..0.0568 rows=236 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Analysis:
As seen above, the only indexing choice that affects query performance is indexing the subnational_data.subnational1 attribute. This is likely because the query has an aggregate GROUP BY clause that groups all data by subnational1. While we cannot see the cost for the aggregate filtering operations when no indexing is applied, the runtime for those filters is significantly reduced from near 3 seconds to approximately 0.18 seconds. This heavily implies that indexing the table on subnational_data.subnational1 made a positive difference and optimized the query.

Since MySQL would have stored the aggregate filter values in a temporary table and postponed the filtering action to the end, we deduced that indexing leads to better performance. Adding any other indexing did not make any difference, as seen from the query analyses above for the other two attributes. This is likely because the subnational_data.gfw_aboveground_carbon_stocks_2000_Mg_C was being aggregated anyway by the subnational_data.subnational1 anyway and because country_data.country is a small table as is (count = 236), so indexing it doesn't make much difference in cost or runtime.

In the end, however, we did not implement any indexing for this query for reasons we explain in query 4: adding indexing to subnational_data.subnational1 drastically worsens the cost of query 4 for a comparatively small improvement here.

Query#4 Identify subnational areas with primary loss above national average

```
mysql> SELECT s.country,
    ->        s.subnational1,
    ->        SUM(s.primary_loss_ha_2023) AS subnational_primary_loss
    -> FROM subnational_data s
    -> JOIN country_data c ON s.country = c.country
    -> GROUP BY s.country, s.subnational1
    -> HAVING SUM(s.primary_loss_ha_2023) > (
    ->     SELECT AVG(primary_loss_ha_2023)
    ->     FROM country_data
    -> )
    -> LIMIT 15;
+---------------------------------+-------------------+--------------------------+
| country                         | subnational1      | subnational_primary_loss |
+---------------------------------+-------------------+--------------------------+
| Bolivia                         | Santa Cruz        |                   222330 |
| Bolivia                         | El Beni           |                   209828 |
| Brazil                          | Roraima           |                    39106 |
| Brazil                          | Rondônia          |                    78023 |
| Brazil                          | Pará              |                   406558 |
| Brazil                          | Mato Grosso       |                   218948 |
| Brazil                          | Amazonas          |                   254890 |
| Brazil                          | Acre              |                    51991 |
| Democratic Republic of the Congo | Tshopo            |                    62261 |
| Democratic Republic of the Congo | Sankuru           |                    40110 |
| Democratic Republic of the Congo | Maï-Ndombe        |                    38634 |
| Indonesia                       | Kalimantan Timur  |                    49037 |
| Indonesia                       | Kalimantan Tengah |                    44073 |
| Indonesia                       | Kalimantan Barat  |                    41216 |
| Madagascar                      | Toamasina         |                    43311 |
+---------------------------------+-------------------+--------------------------+
15 rows in set (0.00 sec)
```

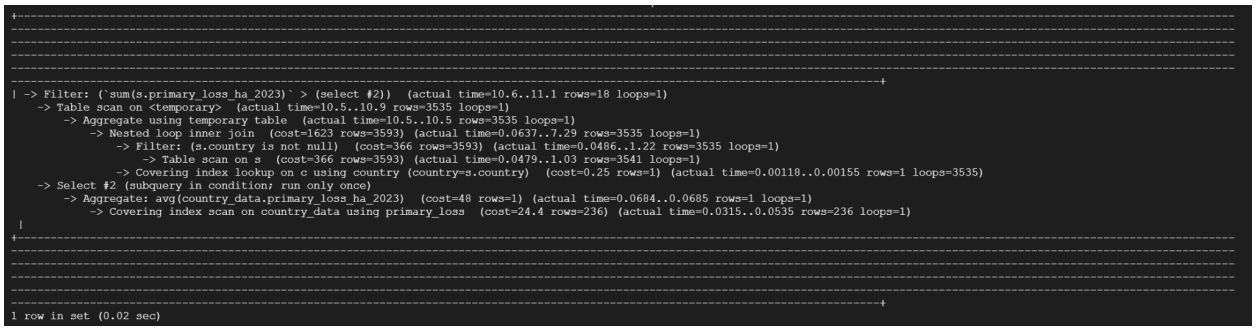
```
SELECT s.country,
      s.subnational1,
      SUM(s.primary_loss_ha_2023) AS subnational_primary_loss
FROM subnational_data s
JOIN country_data c ON s.country = c.country
GROUP BY s.country, s.subnational1
HAVING SUM(s.primary_loss_ha_2023) > (
   SELECT AVG(primary_loss_ha_2023)
   FROM country_data
);
```

No Index:

```
-> Filter: (`sum(s.primary_loss_ha_2023)` > (select #2))  (actual time=5.56..6.14 rows=18 loops=1)
    -> Table scan on <temporary>  (actual time=5.49..5.86 rows=3535 loops=1)
        -> Aggregate using temporary table  (actual time=5.49..5.49 rows=3535 loops=1)
            -> Filter: (s.country = c.country)  (cost=84832 rows=84795) (actual time=0.144..2.24 rows=3535 loops=1)
                -> Inner hash join (<hash>(s.country)=<hash>(c.country))  (cost=84832 rows=84795) (actual time=0.142..1.62 rows=3535 loops=1)
                    -> Table scan on s  (cost=0.205 rows=3593) (actual time=0.0233..0.933 rows=3541 loops=1)
                    -> Hash
                        -> Table scan on c  (cost=24.4 rows=236) (actual time=0.0349..0.0713 rows=236 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(country_data.primary_loss_ha_2023)  (cost=48 rows=1) (actual time=0.0601..0.0601 rows=1 loops=1)
            -> Covering index scan on country_data using primary_loss  (cost=24.4 rows=236) (actual time=0.0239..0.0458 rows=236 loops=1)
1 row in set (0.01 sec)
```

Index on country_data.country:

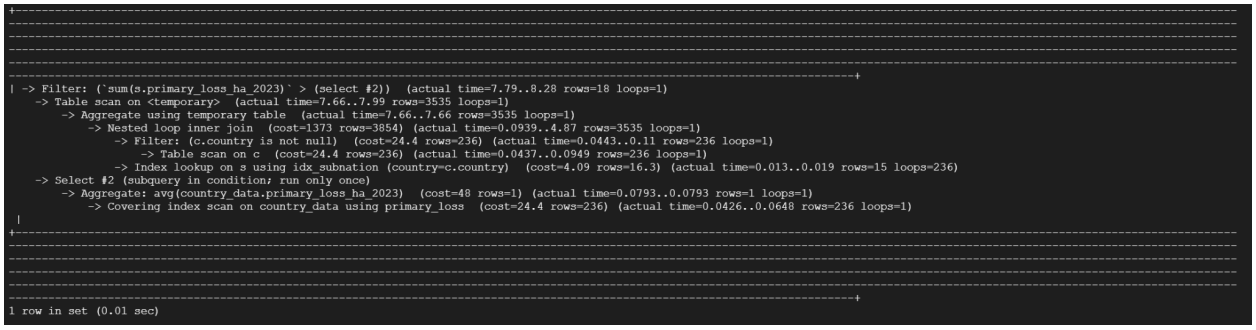
CREATE INDEX idx_country on country_data (country)



```
| -> Filter: (`sum(s.primary_loss_ha_2023)` > (select #2))  (actual time=10.6..11.1 rows=18 loops=1)
    -> Table scan on <temporary>  (actual time=10.5..10.9 rows=3535 loops=1)
        -> Aggregate using temporary table  (actual time=10.5..10.5 rows=3535 loops=1)
            -> Nested loop inner join  (cost=1623 rows=3593) (actual time=0.0637..7.29 rows=3535 loops=1)
                -> Filter: (s.country is not null)  (cost=366 rows=3593) (actual time=0.0486..1.22 rows=3535 loops=1)
                    -> Table scan on s  (cost=366 rows=3593) (actual time=0.0479..1.03 rows=3541 loops=1)
                -> Covering index lookup on c using country (country=s.country)  (cost=0.25 rows=1) (actual time=0.00118..0.00155 rows=1 loops=3535)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(country_data.primary_loss_ha_2023)  (cost=48 rows=1) (actual time=0.0684..0.0685 rows=1 loops=1)
            -> Covering index scan on country_data using primary_loss  (cost=24.4 rows=236) (actual time=0.0315..0.0535 rows=236 loops=1)
 |
1 row in set (0.02 sec)
```

Index on subnational_data.country:

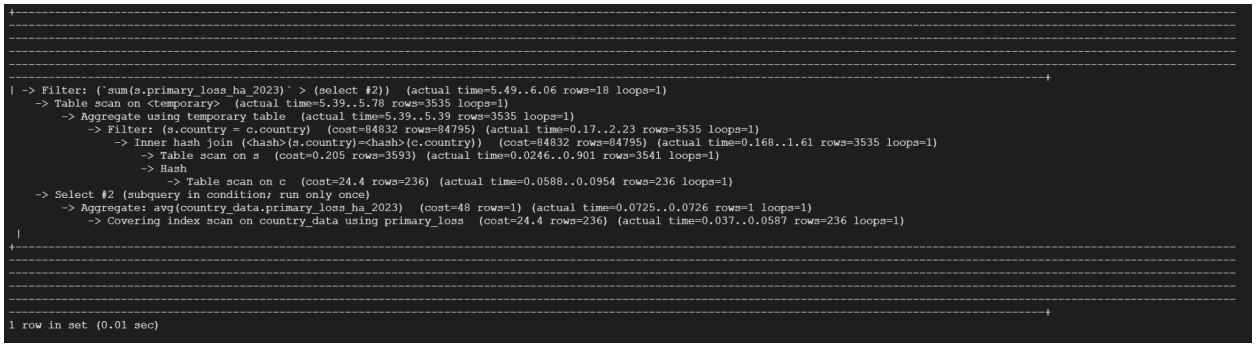
CREATE INDEX idx_subcountry on subnational_data (country)



```
| -> Filter: (`sum(s.primary_loss_ha_2023)` > (select #2))  (actual time=7.79..8.28 rows=18 loops=1)
    -> Table scan on <temporary>  (actual time=7.66..7.99 rows=3535 loops=1)
        -> Aggregate using temporary table  (actual time=7.66..7.66 rows=3535 loops=1)
            -> Nested loop inner join  (cost=1373 rows=3854) (actual time=0.0939..4.87 rows=3535 loops=1)
                -> Filter: (c.country is not null)  (cost=24.4 rows=236) (actual time=0.0443..0.11 rows=236 loops=1)
                    -> Table scan on c  (cost=24.4 rows=236) (actual time=0.0437..0.0949 rows=236 loops=1)
                -> Index lookup on s using idx_subnation (country=c.country)  (cost=4.09 rows=16.3) (actual time=0.013..0.019 rows=15 loops=236)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(country_data.primary_loss_ha_2023)  (cost=48 rows=1) (actual time=0.0793..0.0793 rows=1 loops=1)
            -> Covering index scan on country_data using primary_loss  (cost=24.4 rows=236) (actual time=0.0426..0.0648 rows=236 loops=1)
 |
1 row in set (0.01 sec)
```

Index on subnational_data.subnational1:

CREATE INDEX idx_subnation on subnational_data (subnational1)



```
| -> Filter: (`sum(s.primary_loss_ha_2023)` > (select #2))  (actual time=5.49..6.06 rows=18 loops=1)
    -> Table scan on <temporary>  (actual time=5.39..5.78 rows=3535 loops=1)
        -> Aggregate using temporary table  (actual time=5.39..5.39 rows=3535 loops=1)
            -> Filter: (s.country = c.country)  (cost=84832 rows=84795) (actual time=0.17..2.23 rows=3535 loops=1)
                -> Inner hash join (<hash>(s.country)=<hash>(c.country))  (cost=84832 rows=84795) (actual time=0.168..1.61 rows=3535 loops=1)
                    -> Table scan on s  (cost=0.205 rows=3593) (actual time=0.0246..0.901 rows=3541 loops=1)
                    -> Hash
                        -> Table scan on c  (cost=24.4 rows=236) (actual time=0.0588..0.0954 rows=236 loops=1)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(country_data.primary_loss_ha_2023)  (cost=48 rows=1) (actual time=0.0725..0.0726 rows=1 loops=1)
            -> Covering index scan on country_data using primary_loss  (cost=24.4 rows=236) (actual time=0.037..0.0587 rows=236 loops=1)
 |
1 row in set (0.01 sec)
```

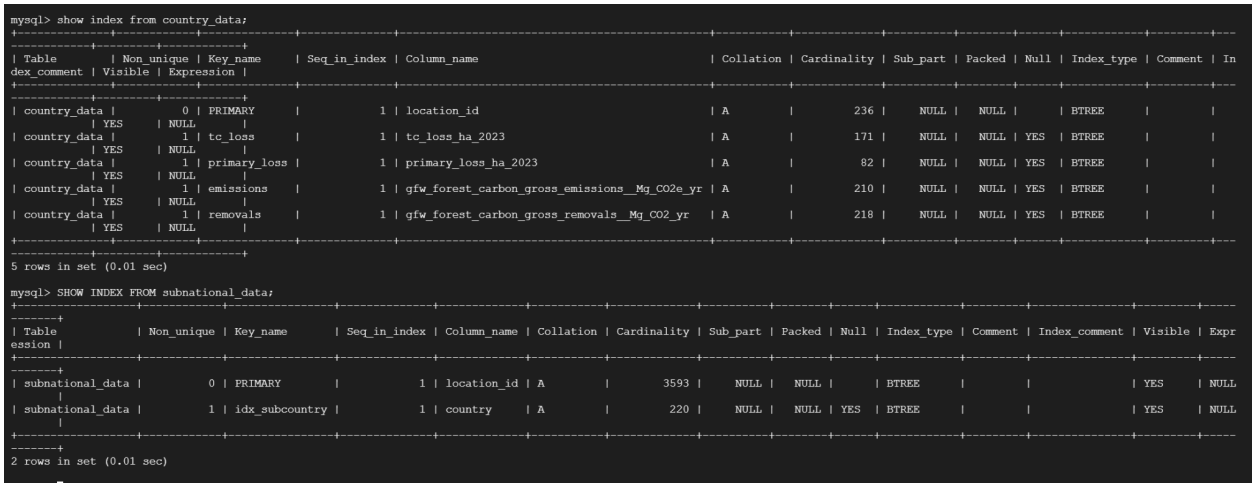
Analysis:

Adding indexing based on country_data.country improved performance, as did adding indexing based on subnational_data.country. Ultimately, we can see that indexing on subnational_data.country led to a greater decrease in cost for the JOIN clause. This is most likely because subnational_data is a much larger table, which leads to greater efficiency than indexing the comparatively small country_data.

Adding indexing based on subnational_data.subnational1 did not improve performance. This is likely because most of the cost for this query goes into the JOIN clause, which does not involve this attribute.

Final Indexing Choices

These are the final indexing choices we made using the explanation for each query tab along with the advanced query we used.



```
mysql> show index from country_data;
+--------------+------------+--------------+--------------+-----------------------------------------------+-----------+-------------+----------+--------+------+------------+---------+---
------------+---------+------------+
| Table        | Non_unique | Key_name     | Seq_in_index | Column_name                                   | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | In
dex_comment | Visible | Expression |
+--------------+------------+--------------+--------------+-----------------------------------------------+-----------+-------------+----------+--------+------+------------+---------+---
------------+---------+------------+
| country_data |          0 | PRIMARY      |            1 | location_id                                   | A         |         236 |     NULL |   NULL |      | BTREE      |         |
            | YES     | NULL       |
| country_data |          1 | tc_loss      |            1 | tc_loss_ha_2023                               | A         |         171 |     NULL |   NULL | YES  | BTREE      |         |
            | YES     | NULL       |
| country_data |          1 | primary_loss |            1 | primary_loss_ha_2023                          | A         |          82 |     NULL |   NULL | YES  | BTREE      |         |
            | YES     | NULL       |
| country_data |          1 | emissions    |            1 | gfw_forest_carbon_gross_emissions__Mg_CO2e_yr | A         |         210 |     NULL |   NULL | YES  | BTREE      |         |
            | YES     | NULL       |
| country_data |          1 | removals     |            1 | gfw_forest_carbon_gross_removals__Mg_CO2_yr   | A         |         218 |     NULL |   NULL | YES  | BTREE      |         |
            | YES     | NULL       |
+--------------+------------+--------------+--------------+-----------------------------------------------+-----------+-------------+----------+--------+------+------------+---------+---
------------+---------+------------+
5 rows in set (0.01 sec)

mysql> SHOW INDEX FROM subnational_data;
+------------------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+-----
-------+
| Table            | Non_unique | Key_name       | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment | Visible | Expr
ession |
+------------------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+-----
-------+
| subnational_data |          0 | PRIMARY        |            1 | location_id | A         |        3593 |     NULL |   NULL |      | BTREE      |         |               | YES     | NULL
       |
| subnational_data |          1 | idx_subcountry |            1 | country     | A         |         220 |     NULL |   NULL | YES  | BTREE      |         |               | YES     | NULL
       |
+------------------+------------+----------------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+---------+-----
-------+
2 rows in set (0.01 sec)
```