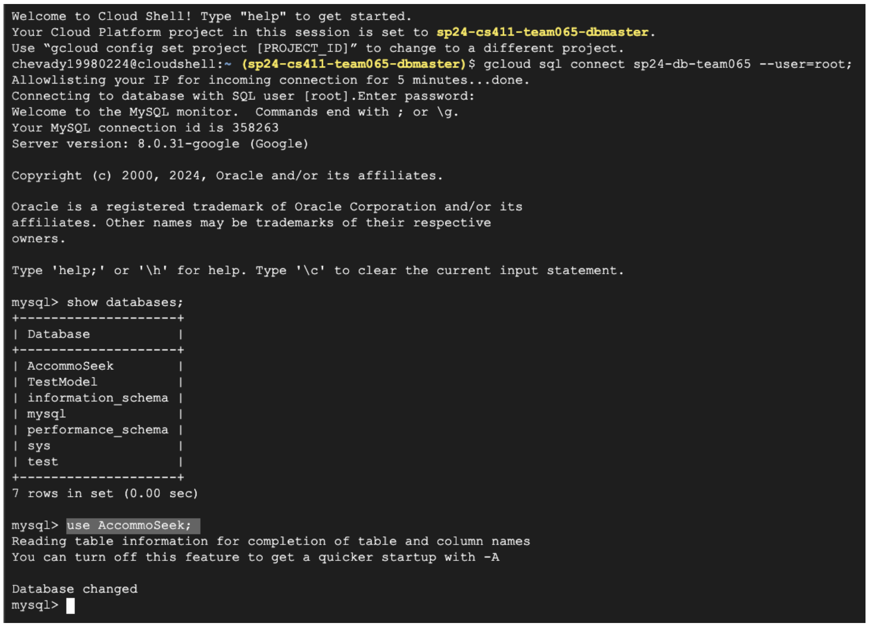
Database Implementation

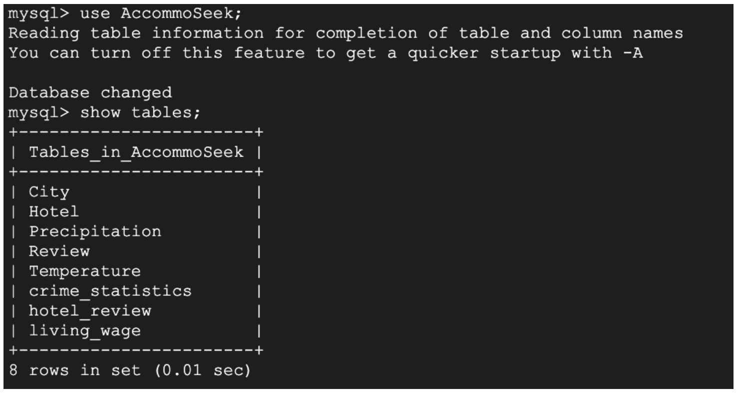
Database Implementation on GCP

Connecting to GCP:

gcloud sql connect sp24-db-team065 --user=root;
show databases;
use AccommoSeek;
show tables;



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to sp24-cs411-team065-dbmaster.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
chevady19980224@cloudshell:~ (sp24-cs411-team065-dbmaster)$ gcloud sql connect sp24-db-team065 --user=root;
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 358263
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           |
+--------------------+
| AccommoSeek        |
| TestModel          |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| test               |
+--------------------+
7 rows in set (0.00 sec)

mysql> use AccommoSeek;
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
mysql> 
```



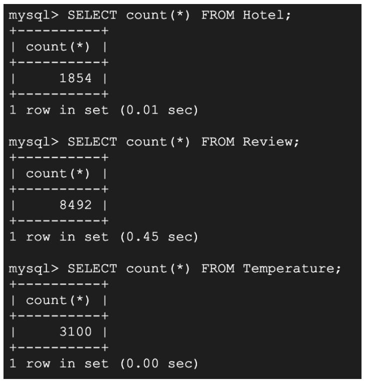
```
mysql> use AccommoSeek;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------------+
| Tables_in_AccommoSeek |
+-----------------------+
| City                  |
| Hotel                 |
| Precipitation         |
| Review                |
| Temperature           |
| crime_statistics      |
| hotel_review          |
| living_wage           |
+-----------------------+
8 rows in set (0.01 sec)
```

crime_statistics, hotel_review, living_wage tables are used as temporary tables to clean and export data into City and Review table.

Data Definition Language:

```
CREATE TABLE City (
  Name VARCHAR(255) PRIMARY KEY,
  State VARCHAR(255),
  LivingWage REAL,
  Population INT,
  CrimeFrequency REAL
);

CREATE TABLE Hotel (
  Name VARCHAR(255),
  Address VARCHAR(255),
  CityName VARCHAR(255),
  PRIMARY KEY (Name, CityName),
  FOREIGN KEY (CityName) REFERENCES City(Name)
);

CREATE TABLE Review (
  UserName VARCHAR(255),
  Title VARCHAR(255),
  Text VARCHAR(1024),
  Rating REAL,
  Date datetime,
  HotelName VARCHAR(255),
  CityName VARCHAR(255),
  PRIMARY KEY (UserName, HotelName, CityName),
  FOREIGN KEY (HotelName) REFERENCES Hotel(Name),
  FOREIGN KEY (CityName) REFERENCES City(Name)
);

CREATE TABLE Temperature (
  Month INT,
  Year INT,
  Temperature REAL,
  CityName VARCHAR(255),
  PRIMARY KEY (Month, Year, CityName),
  FOREIGN KEY (CityName) REFERENCES City(Name)
);

CREATE TABLE Precipitation(
  CityState VARCHAR(255),
  Month INT,
  Precipitation REAL,
  PRIMARY KEY (CityState, Month),
  FOREIGN KEY (CityState) REFERENCES City(State)
);
```

Inserting Data



```
mysql> SELECT count(*) FROM Hotel;
+----------+
| count(*) |
+----------+
|     1854 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT count(*) FROM Review;
+----------+
| count(*) |
+----------+
|     8492 |
+----------+
1 row in set (0.45 sec)

mysql> SELECT count(*) FROM Temperature;
+----------+
| count(*) |
+----------+
|     3100 |
+----------+
1 row in set (0.00 sec)
```

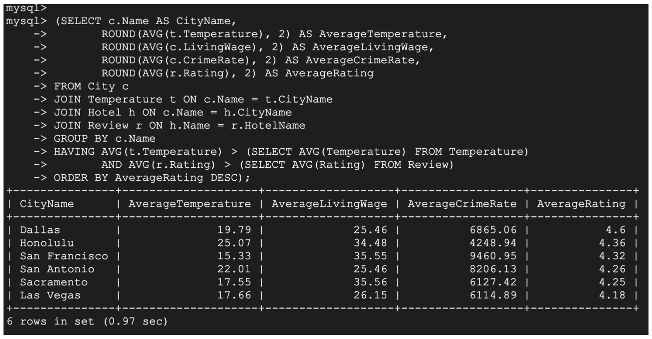
Advanced Queries

Advanced Query 1

Advanced features: join multiple relations, aggregation via GROUP BY, subqueries

```
SELECT
        c.Name AS CityName,
        ROUND(AVG(t.Temperature), 2) AS AverageTemperature,
        ROUND(AVG(c.LivingWage), 2) AS AverageLivingWage,
        ROUND(AVG(c.CrimeRate), 2) AS AverageCrimeRate,
        ROUND(AVG(r.Rating), 2) AS AverageRating
FROM City c
        JOIN Temperature t ON c.Name = t.CityName
        JOIN Hotel h ON c.Name = h.CityName
        JOIN Review r ON h.Name = r.HotelName
GROUP BY c.Name
HAVING
        AVG(t.Temperature) > (SELECT AVG(Temperature) FROM Temperature)
        AND AVG(r.Rating) > (SELECT AVG(Rating) FROM Review)
ORDER BY AverageRating DESC;
```

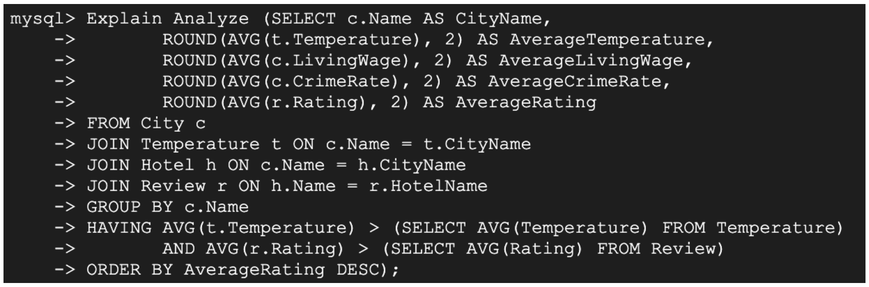
Top 15 results:



```
mysql>
mysql> (SELECT c.Name AS CityName,
    ->        ROUND(AVG(t.Temperature), 2) AS AverageTemperature,
    ->        ROUND(AVG(c.LivingWage), 2) AS AverageLivingWage,
    ->        ROUND(AVG(c.CrimeRate), 2) AS AverageCrimeRate,
    ->        ROUND(AVG(r.Rating), 2) AS AverageRating
    -> FROM City c
    -> JOIN Temperature t ON c.Name = t.CityName
    -> JOIN Hotel h ON c.Name = h.CityName
    -> JOIN Review r ON h.Name = r.HotelName
    -> GROUP BY c.Name
    -> HAVING AVG(t.Temperature) > (SELECT AVG(Temperature) FROM Temperature)
    ->        AND AVG(r.Rating) > (SELECT AVG(Rating) FROM Review)
    -> ORDER BY AverageRating DESC);
+---------------+--------------------+-------------------+------------------+---------------+
| CityName      | AverageTemperature | AverageLivingWage | AverageCrimeRate | AverageRating |
+---------------+--------------------+-------------------+------------------+---------------+
| Dallas        |              19.79 |             25.46 |          6865.06 |           4.6 |
| Honolulu      |              25.07 |             34.48 |          4248.94 |          4.36 |
| San Francisco |              15.33 |             35.55 |          9460.95 |          4.32 |
| San Antonio   |              22.01 |             25.46 |          8206.13 |          4.26 |
| Sacramento    |              17.55 |             35.56 |          6127.42 |          4.25 |
| Las Vegas     |              17.66 |             26.15 |          6114.89 |          4.18 |
+---------------+--------------------+-------------------+------------------+---------------+
6 rows in set (0.97 sec)
```

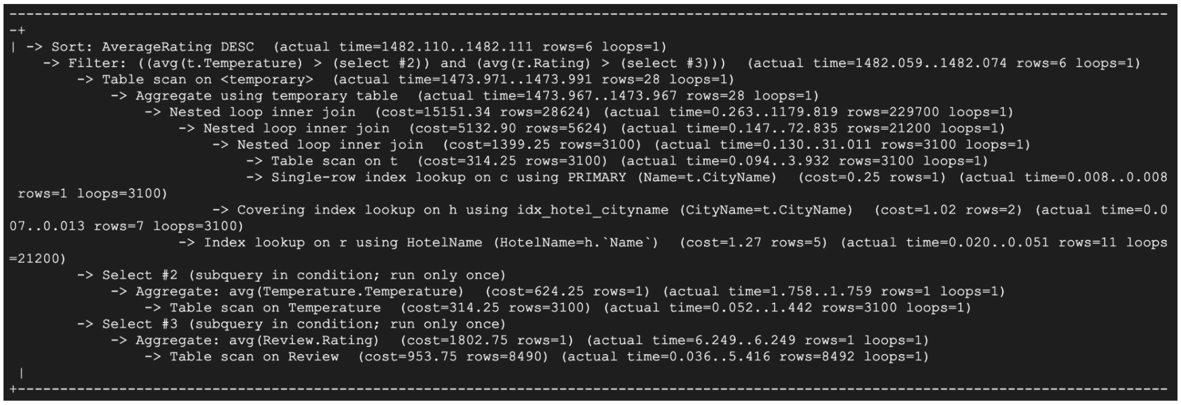
The output of the result has only 6 rows

Command:



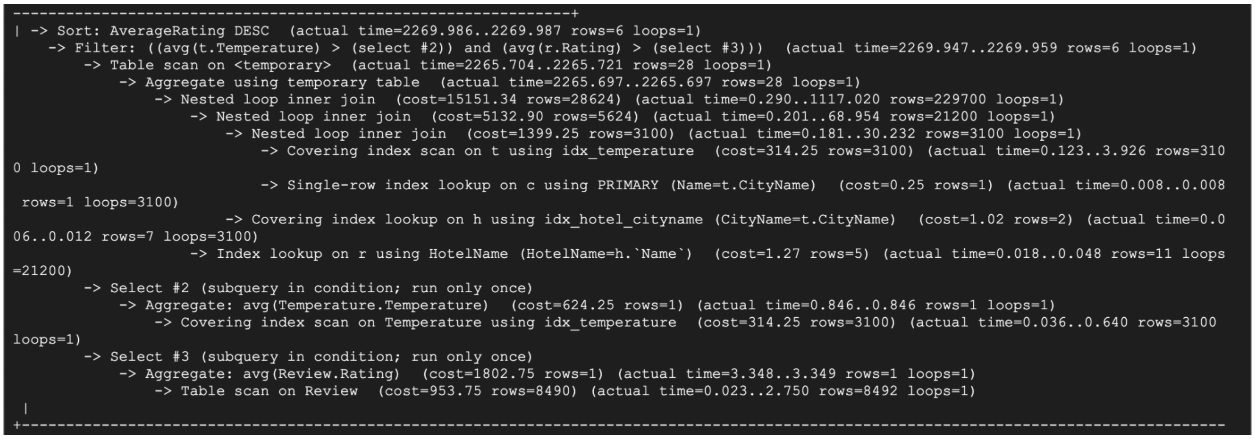
```
mysql> Explain Analyze (SELECT c.Name AS CityName,
    ->        ROUND(AVG(t.Temperature), 2) AS AverageTemperature,
    ->        ROUND(AVG(c.LivingWage), 2) AS AverageLivingWage,
    ->        ROUND(AVG(c.CrimeRate), 2) AS AverageCrimeRate,
    ->        ROUND(AVG(r.Rating), 2) AS AverageRating
    -> FROM City c
    -> JOIN Temperature t ON c.Name = t.CityName
    -> JOIN Hotel h ON c.Name = h.CityName
    -> JOIN Review r ON h.Name = r.HotelName
    -> GROUP BY c.Name
    -> HAVING AVG(t.Temperature) > (SELECT AVG(Temperature) FROM Temperature)
    ->        AND AVG(r.Rating) > (SELECT AVG(Rating) FROM Review)
    -> ORDER BY AverageRating DESC);
```

Analysis before indexing:



```
-------------------------------------------------------------------------------------------------------------------------------------------
-+
| -> Sort: AverageRating DESC  (actual time=1482.110..1482.111 rows=6 loops=1)
    -> Filter: ((avg(t.Temperature) > (select #2)) and (avg(r.Rating) > (select #3)))  (actual time=1482.059..1482.074 rows=6 loops=1)
        -> Table scan on <temporary>  (actual time=1473.971..1473.991 rows=28 loops=1)
            -> Aggregate using temporary table  (actual time=1473.967..1473.967 rows=28 loops=1)
                -> Nested loop inner join  (cost=15151.34 rows=28624) (actual time=0.263..1179.819 rows=229700 loops=1)
                    -> Nested loop inner join  (cost=5132.90 rows=5624) (actual time=0.147..72.835 rows=21200 loops=1)
                        -> Nested loop inner join  (cost=1399.25 rows=3100) (actual time=0.130..31.011 rows=3100 loops=1)
                            -> Table scan on t  (cost=314.25 rows=3100) (actual time=0.094..3.932 rows=3100 loops=1)
                            -> Single-row index lookup on c using PRIMARY (Name=t.CityName)  (cost=0.25 rows=1) (actual time=0.008..0.008
 rows=1 loops=3100)
                        -> Covering index lookup on h using idx_hotel_cityname (CityName=t.CityName)  (cost=1.02 rows=2) (actual time=0.0
07..0.013 rows=7 loops=3100)
                    -> Index lookup on r using HotelName (HotelName=h.`Name`)  (cost=1.27 rows=5) (actual time=0.020..0.051 rows=11 loops
=21200)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(Temperature.Temperature)  (cost=624.25 rows=1) (actual time=1.758..1.759 rows=1 loops=1)
                -> Table scan on Temperature  (cost=314.25 rows=3100) (actual time=0.052..1.442 rows=3100 loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(Review.Rating)  (cost=1802.75 rows=1) (actual time=6.249..6.249 rows=1 loops=1)
                -> Table scan on Review  (cost=953.75 rows=8490) (actual time=0.036..5.416 rows=8492 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------
```

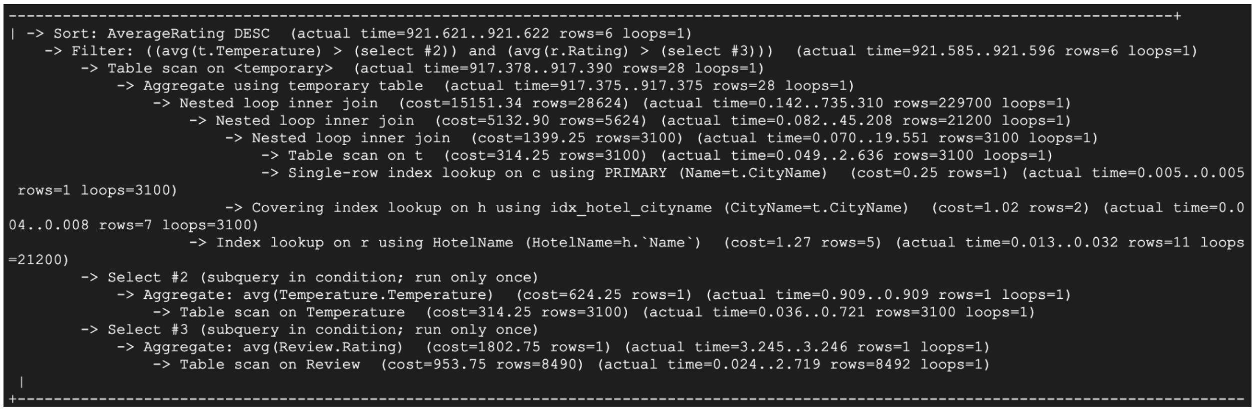
Attempt 1 : Index on Temperature



```
---------------------------------------------------------------+
| -> Sort: AverageRating DESC  (actual time=2269.986..2269.987 rows=6 loops=1)
    -> Filter: ((avg(t.Temperature) > (select #2)) and (avg(r.Rating) > (select #3)))  (actual time=2269.947..2269.959 rows=6 loops=1)
        -> Table scan on <temporary>  (actual time=2265.704..2265.721 rows=28 loops=1)
            -> Aggregate using temporary table  (actual time=2265.697..2265.697 rows=28 loops=1)
                -> Nested loop inner join  (cost=15151.34 rows=28624) (actual time=0.290..1117.020 rows=229700 loops=1)
                    -> Nested loop inner join  (cost=5132.90 rows=5624) (actual time=0.201..68.954 rows=21200 loops=1)
                        -> Nested loop inner join  (cost=1399.25 rows=3100) (actual time=0.181..30.232 rows=3100 loops=1)
                            -> Covering index scan on t using idx_temperature  (cost=314.25 rows=3100) (actual time=0.123..3.926 rows=310
0 loops=1)
                            -> Single-row index lookup on c using PRIMARY (Name=t.CityName)  (cost=0.25 rows=1) (actual time=0.008..0.008
 rows=1 loops=3100)
                        -> Covering index lookup on h using idx_hotel_cityname (CityName=t.CityName)  (cost=1.02 rows=2) (actual time=0.0
06..0.012 rows=7 loops=3100)
                    -> Index lookup on r using HotelName (HotelName=h.`Name`)  (cost=1.27 rows=5) (actual time=0.018..0.048 rows=11 loops
=21200)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(Temperature.Temperature)  (cost=624.25 rows=1) (actual time=0.846..0.846 rows=1 loops=1)
                -> Covering index scan on Temperature using idx_temperature  (cost=314.25 rows=3100) (actual time=0.036..0.640 rows=3100
loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(Review.Rating)  (cost=1802.75 rows=1) (actual time=3.348..3.349 rows=1 loops=1)
                -> Table scan on Review  (cost=953.75 rows=8490) (actual time=0.023..2.750 rows=8492 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------------------------
```

We expected to improve performance by indexing attributes in the HAVING clause, like 'Temperature'. The rationale behind this approach is that indexing should narrow down the search space to only include records meeting the specified condition, thereby improving efficiency. However, we've not seen a reduction in query costs. Specifically, even as the scan on 'Temperature' shifts from a table scan to an index scan, the cost remains static at 314.25. We suspect the reason is that our table's size is insufficient to exhibit significant improvements when weighed against the overhead of indexing in RDBMS.

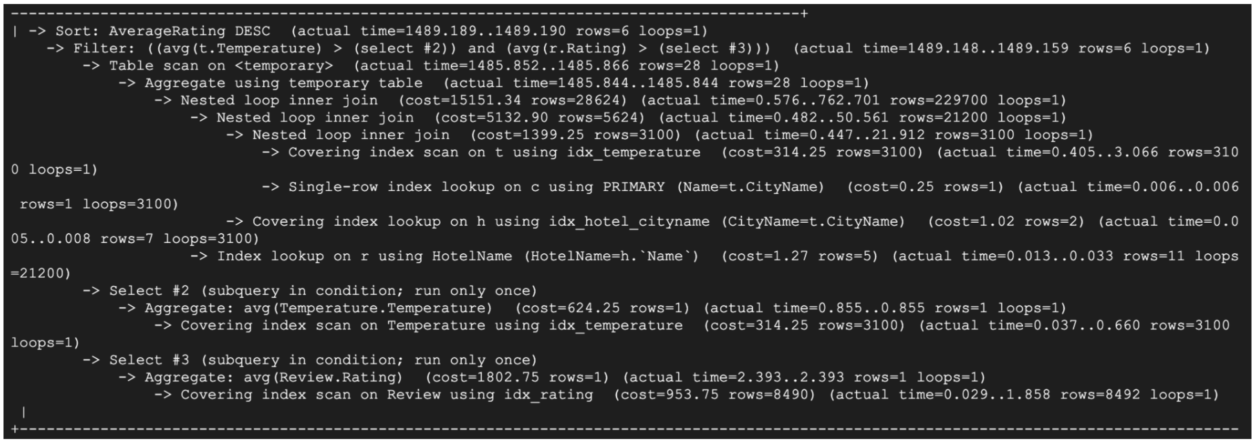
Attempt 2 : Index on CrimeRate



```
-------------------------------------------------------------------------------------------------------------------------------+
| -> Sort: AverageRating DESC  (actual time=921.621..921.622 rows=6 loops=1)
    -> Filter: ((avg(t.Temperature) > (select #2)) and (avg(r.Rating) > (select #3)))  (actual time=921.585..921.596 rows=6 loops=1)
        -> Table scan on <temporary>  (actual time=917.378..917.390 rows=28 loops=1)
            -> Aggregate using temporary table  (actual time=917.375..917.375 rows=28 loops=1)
                -> Nested loop inner join  (cost=15151.34 rows=28624) (actual time=0.142..735.310 rows=229700 loops=1)
                    -> Nested loop inner join  (cost=5132.90 rows=5624) (actual time=0.082..45.208 rows=21200 loops=1)
                        -> Nested loop inner join  (cost=1399.25 rows=3100) (actual time=0.070..19.551 rows=3100 loops=1)
                            -> Table scan on t  (cost=314.25 rows=3100) (actual time=0.049..2.636 rows=3100 loops=1)
                            -> Single-row index lookup on c using PRIMARY (Name=t.CityName)  (cost=0.25 rows=1) (actual time=0.005..0.005
 rows=1 loops=3100)
                        -> Covering index lookup on h using idx_hotel_cityname (CityName=t.CityName)  (cost=1.02 rows=2) (actual time=0.0
04..0.008 rows=7 loops=3100)
                    -> Index lookup on r using HotelName (HotelName=h.`Name`)  (cost=1.27 rows=5) (actual time=0.013..0.032 rows=11 loops
=21200)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(Temperature.Temperature)  (cost=624.25 rows=1) (actual time=0.909..0.909 rows=1 loops=1)
                -> Table scan on Temperature  (cost=314.25 rows=3100) (actual time=0.036..0.721 rows=3100 loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(Review.Rating)  (cost=1802.75 rows=1) (actual time=3.245..3.246 rows=1 loops=1)
                -> Table scan on Review  (cost=953.75 rows=8490) (actual time=0.024..2.719 rows=8492 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------
```

The addition of an index on the attribute present in the SELECT clause, specifically 'CrimeRate,' was not expected to influence the cost. The outcomes of our analysis align with these expectations, confirming that the index has no impact on query cost.

Attempt 3 : Index on Temperature and Rating



```
-------------------------------------------------------------------------------------+
| -> Sort: AverageRating DESC  (actual time=1489.189..1489.190 rows=6 loops=1)
    -> Filter: ((avg(t.Temperature) > (select #2)) and (avg(r.Rating) > (select #3)))  (actual time=1489.148..1489.159 rows=6 loops=1)
        -> Table scan on <temporary>  (actual time=1485.852..1485.866 rows=28 loops=1)
            -> Aggregate using temporary table  (actual time=1485.844..1485.844 rows=28 loops=1)
                -> Nested loop inner join  (cost=15151.34 rows=28624) (actual time=0.576..762.701 rows=229700 loops=1)
                    -> Nested loop inner join  (cost=5132.90 rows=5624) (actual time=0.482..50.561 rows=21200 loops=1)
                        -> Nested loop inner join  (cost=1399.25 rows=3100) (actual time=0.447..21.912 rows=3100 loops=1)
                            -> Covering index scan on t using idx_temperature  (cost=314.25 rows=3100) (actual time=0.405..3.066 rows=310
0 loops=1)
                            -> Single-row index lookup on c using PRIMARY (Name=t.CityName)  (cost=0.25 rows=1) (actual time=0.006..0.006
 rows=1 loops=3100)
                        -> Covering index lookup on h using idx_hotel_cityname (CityName=t.CityName)  (cost=1.02 rows=2) (actual time=0.0
05..0.008 rows=7 loops=3100)
                    -> Index lookup on r using HotelName (HotelName=h.`Name`)  (cost=1.27 rows=5) (actual time=0.013..0.033 rows=11 loops
=21200)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(Temperature.Temperature)  (cost=624.25 rows=1) (actual time=0.855..0.855 rows=1 loops=1)
                -> Covering index scan on Temperature using idx_temperature  (cost=314.25 rows=3100) (actual time=0.037..0.660 rows=3100
loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(Review.Rating)  (cost=1802.75 rows=1) (actual time=2.393..2.393 rows=1 loops=1)
                -> Covering index scan on Review using idx_rating  (cost=953.75 rows=8490) (actual time=0.029..1.858 rows=8492 loops=1)
 |
+--------------------------------------------------------------------------------------------------------------------------------
```

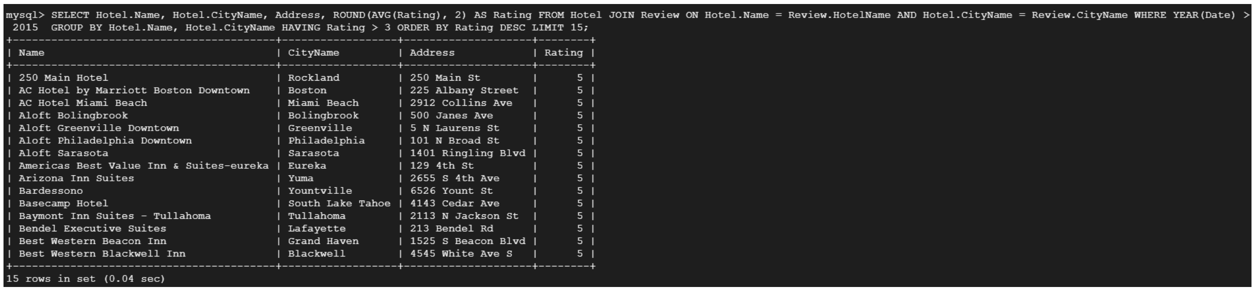
Likewise, we anticipate enhancing performance by adding an index to attributes present in the HAVING clause, such as the attribute 'Temperature' and ‘Rating’ in this context. However, we have not observed a reduction in query cost. Similarly, we believe that the size of our table may not be substantial enough to exhibit significant improvements.

Advanced Query 2

Advanced features: join multiple relations, aggregation via GROUP BY

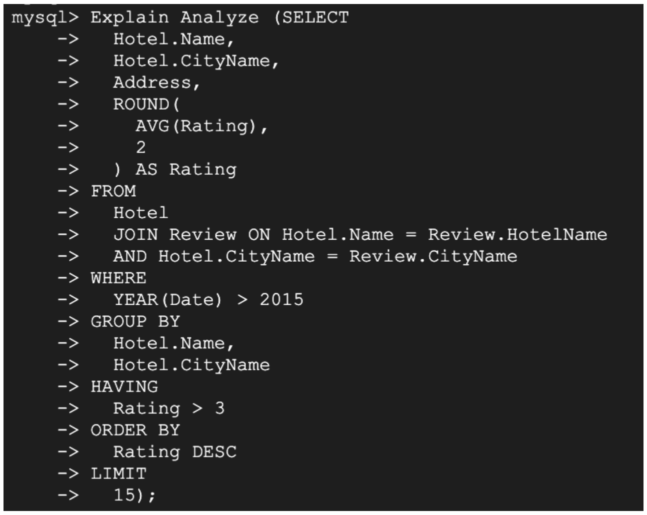
```
SELECT
        Hotel.Name,
        Hotel.CityName,
        Address,
        ROUND(AVG(Rating), 2) AS Rating
FROM Hotel
        JOIN Review ON Hotel.Name = Review.HotelName AND Hotel.CityName =
Review.CityName WHERE YEAR(Date) > 2015
GROUP BY
        Hotel.Name,
        Hotel.CityName
HAVING Rating > 3
ORDER BY Rating DESC
LIMIT 15;
```

Top 15 results:



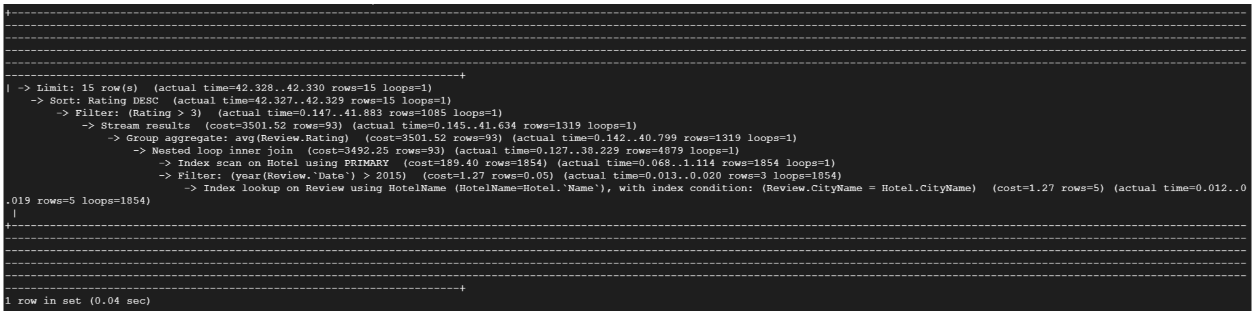
```
mysql> SELECT Hotel.Name, Hotel.CityName, Address, ROUND(AVG(Rating), 2) AS Rating FROM Hotel JOIN Review ON Hotel.Name = Review.HotelName AND Hotel.CityName = Review.CityName WHERE YEAR(Date) >
 2015  GROUP BY Hotel.Name, Hotel.CityName HAVING Rating > 3 ORDER BY Rating DESC LIMIT 15;
+-----------------------------------------+------------------+--------------------+--------+
| Name                                    | CityName         | Address            | Rating |
+-----------------------------------------+------------------+--------------------+--------+
| 250 Main Hotel                          | Rockland         | 250 Main St        |      5 |
| AC Hotel by Marriott Boston Downtown    | Boston           | 225 Albany Street  |      5 |
| AC Hotel Miami Beach                    | Miami Beach      | 2912 Collins Ave   |      5 |
| Aloft Bolingbrook                       | Bolingbrook      | 500 Janes Ave      |      5 |
| Aloft Greenville Downtown               | Greenville       | 5 N Laurens St     |      5 |
| Aloft Philadelphia Downtown             | Philadelphia     | 101 N Broad St     |      5 |
| Aloft Sarasota                          | Sarasota         | 1401 Ringling Blvd |      5 |
| Americas Best Value Inn & Suites-eureka | Eureka           | 129 4th St         |      5 |
| Arizona Inn Suites                      | Yuma             | 2655 S 4th Ave     |      5 |
| Bardessono                              | Yountville       | 6526 Yount St      |      5 |
| Basecamp Hotel                          | South Lake Tahoe | 4143 Cedar Ave     |      5 |
| Baymont Inn Suites - Tullahoma          | Tullahoma        | 2113 N Jackson St  |      5 |
| Bendel Executive Suites                 | Lafayette        | 213 Bendel Rd      |      5 |
| Best Western Beacon Inn                 | Grand Haven      | 1525 S Beacon Blvd |      5 |
| Best Western Blackwell Inn              | Blackwell        | 4545 White Ave S   |      5 |
+-----------------------------------------+------------------+--------------------+--------+
15 rows in set (0.04 sec)
```

Command:



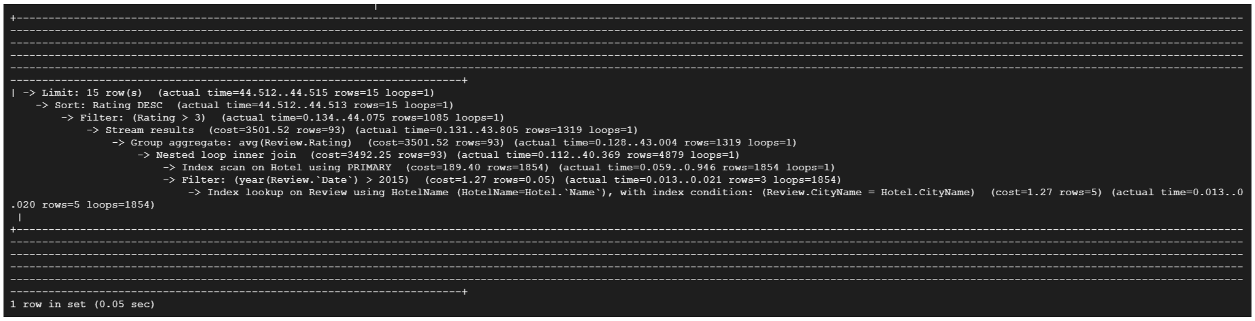
```
mysql> Explain Analyze (SELECT
    ->   Hotel.Name,
    ->   Hotel.CityName,
    ->   Address,
    ->   ROUND(
    ->     AVG(Rating),
    ->     2
    ->   ) AS Rating
    -> FROM
    ->   Hotel
    ->   JOIN Review ON Hotel.Name = Review.HotelName
    ->   AND Hotel.CityName = Review.CityName
    -> WHERE
    ->   YEAR(Date) > 2015
    -> GROUP BY
    ->   Hotel.Name,
    ->   Hotel.CityName
    -> HAVING
    ->   Rating > 3
    -> ORDER BY
    ->   Rating DESC
    -> LIMIT
    ->   15);
```

Analysis before indexing:



```
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=42.328..42.330 rows=15 loops=1)
    -> Sort: Rating DESC  (actual time=42.327..42.329 rows=15 loops=1)
        -> Filter: (Rating > 3)  (actual time=0.147..41.883 rows=1085 loops=1)
            -> Stream results  (cost=3501.52 rows=93) (actual time=0.145..41.634 rows=1319 loops=1)
                -> Group aggregate: avg(Review.Rating)  (cost=3501.52 rows=93) (actual time=0.142..40.799 rows=1319 loops=1)
                    -> Nested loop inner join  (cost=3492.25 rows=93) (actual time=0.127..38.229 rows=4879 loops=1)
                        -> Index scan on Hotel using PRIMARY  (cost=189.40 rows=1854) (actual time=0.068..1.114 rows=1854 loops=1)
                        -> Filter: (year(Review.`Date`) > 2015)  (cost=1.27 rows=0.05) (actual time=0.013..0.020 rows=3 loops=1854)
                            -> Index lookup on Review using HotelName (HotelName=Hotel.`Name`), with index condition: (Review.CityName = Hotel.CityName)  (cost=1.27 rows=5) (actual time=0.012..0
.019 rows=5 loops=1854)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
1 row in set (0.04 sec)
```

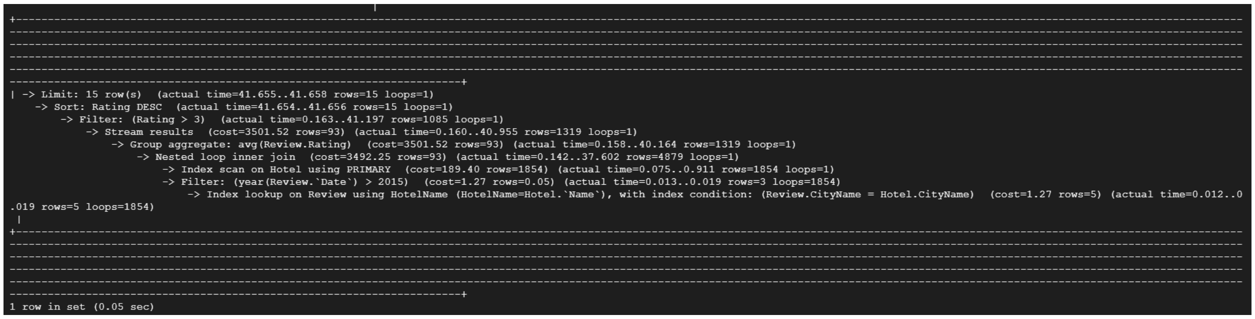
Attempt 1 : Index on Review.Rating



```
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=44.512..44.515 rows=15 loops=1)
    -> Sort: Rating DESC  (actual time=44.512..44.513 rows=15 loops=1)
        -> Filter: (Rating > 3)  (actual time=0.134..44.075 rows=1085 loops=1)
            -> Stream results  (cost=3501.52 rows=93) (actual time=0.131..43.805 rows=1319 loops=1)
                -> Group aggregate: avg(Review.Rating)  (cost=3501.52 rows=93) (actual time=0.128..43.004 rows=1319 loops=1)
                    -> Nested loop inner join  (cost=3492.25 rows=93) (actual time=0.112..40.369 rows=4879 loops=1)
                        -> Index scan on Hotel using PRIMARY  (cost=189.40 rows=1854) (actual time=0.059..0.946 rows=1854 loops=1)
                        -> Filter: (year(Review.`Date`) > 2015)  (cost=1.27 rows=0.05) (actual time=0.013..0.021 rows=3 loops=1854)
                            -> Index lookup on Review using HotelName (HotelName=Hotel.`Name`), with index condition: (Review.CityName = Hotel.CityName)  (cost=1.27 rows=5) (actual time=0.013..0
.020 rows=5 loops=1854)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
1 row in set (0.05 sec)
```

Likewise, we anticipate enhancing performance by adding an index to attributes present in the HAVING clause, such as the attribute 'Rating' in this context. However, we have not observed a reduction in query cost. Similarly, we believe that the size of our table may not be substantial enough to exhibit significant improvements.

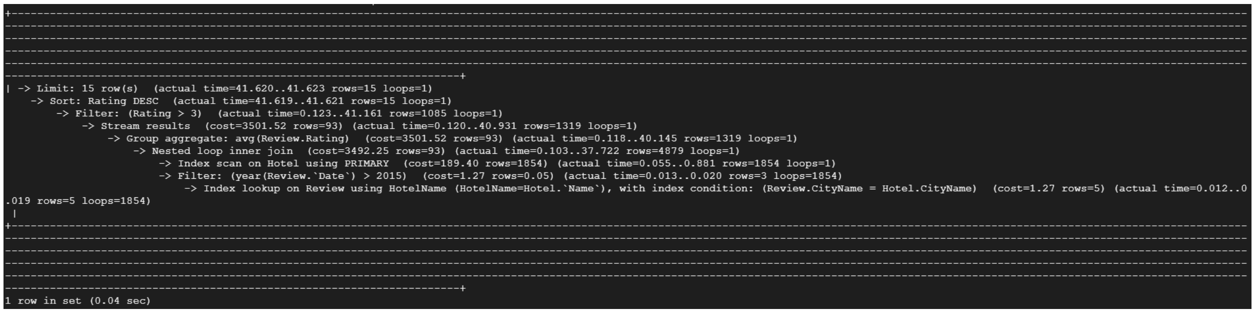
Attempt 2: Index on Review.Date



```
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=41.655..41.658 rows=15 loops=1)
    -> Sort: Rating DESC  (actual time=41.654..41.656 rows=15 loops=1)
        -> Filter: (Rating > 3)  (actual time=0.163..41.197 rows=1085 loops=1)
            -> Stream results  (cost=3501.52 rows=93) (actual time=0.160..40.955 rows=1319 loops=1)
                -> Group aggregate: avg(Review.Rating)  (cost=3501.52 rows=93) (actual time=0.158..40.164 rows=1319 loops=1)
                    -> Nested loop inner join  (cost=3492.25 rows=93) (actual time=0.142..37.602 rows=4879 loops=1)
                        -> Index scan on Hotel using PRIMARY  (cost=189.40 rows=1854) (actual time=0.075..0.911 rows=1854 loops=1)
                        -> Filter: (year(Review.`Date`) > 2015)  (cost=1.27 rows=0.05) (actual time=0.013..0.019 rows=3 loops=1854)
                            -> Index lookup on Review using HotelName (HotelName=Hotel.`Name`), with index condition: (Review.CityName = Hotel.CityName)  (cost=1.27 rows=5) (actual time=0.012..0
.019 rows=5 loops=1854)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------+
1 row in set (0.05 sec)
```

Similarly, we expected to boost performance by indexing attributes mentioned in the WHERE clause, like 'Date'. The logic behind this strategy mirrors that of indexing attributes in the HAVING clause. Despite these efforts, there has been no noticeable reduction in query costs. Once again, it appears that the relatively small size of our table may not allow for the significant improvements we anticipated.

Attempt 3: Composite index on Review.Rating & Review.Date



```
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=41.620..41.623 rows=15 loops=1)
    -> Sort: Rating DESC  (actual time=41.619..41.621 rows=15 loops=1)
        -> Filter: (Rating > 3)  (actual time=0.123..41.161 rows=1085 loops=1)
            -> Stream results  (cost=3501.52 rows=93) (actual time=0.120..40.931 rows=1319 loops=1)
                -> Group aggregate: avg(Review.Rating)  (cost=3501.52 rows=93) (actual time=0.118..40.145 rows=1319 loops=1)
                    -> Nested loop inner join  (cost=3492.25 rows=93) (actual time=0.103..37.722 rows=4879 loops=1)
                        -> Index scan on Hotel using PRIMARY  (cost=189.40 rows=1854) (actual time=0.055..0.881 rows=1854 loops=1)
                        -> Filter: (year(Review.`Date`) > 2015)  (cost=1.27 rows=0.05) (actual time=0.013..0.020 rows=3 loops=1854)
                            -> Index lookup on Review using HotelName (HotelName=Hotel.`Name`), with index condition: (Review.CityName = Hotel.CityName)  (cost=1.27 rows=5) (actual time=0.012..0
.019 rows=5 loops=1854)
 |
+------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------+
1 row in set (0.04 sec)
```

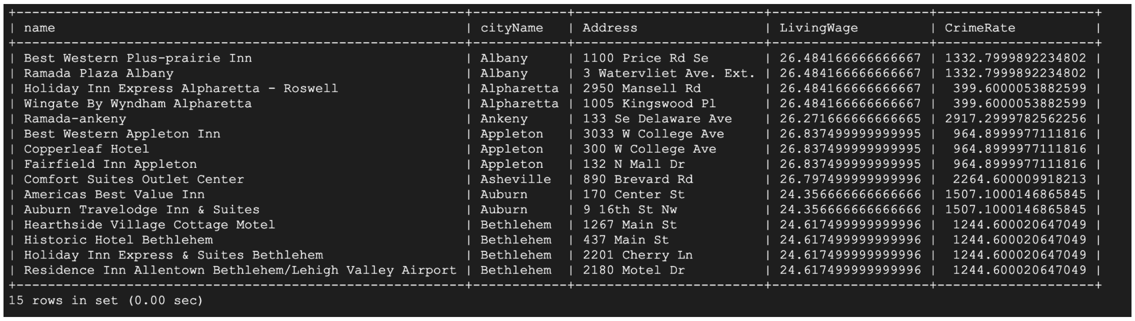
Creating a composite index on 'Rating' and 'Date' has yielded the same results as previously observed.

Advanced Query 3

Advanced features: join multiple relations, subqueries

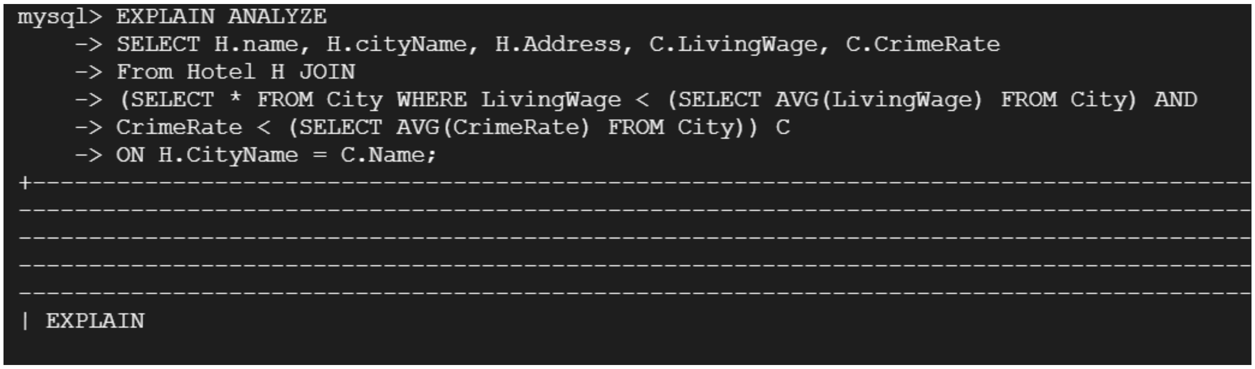
SELECT
        H.name,
        H.cityName,
        H.Address,
        C.LivingWage,
        C.CrimeRate
From Hotel H
JOIN (SELECT * FROM City WHERE LivingWage < (SELECT AVG(LivingWage) FROM City)
         AND CrimeRate < (SELECT AVG(CrimeRate) FROM City)) C ON H.CityName = C.Name;

Top 15 results:



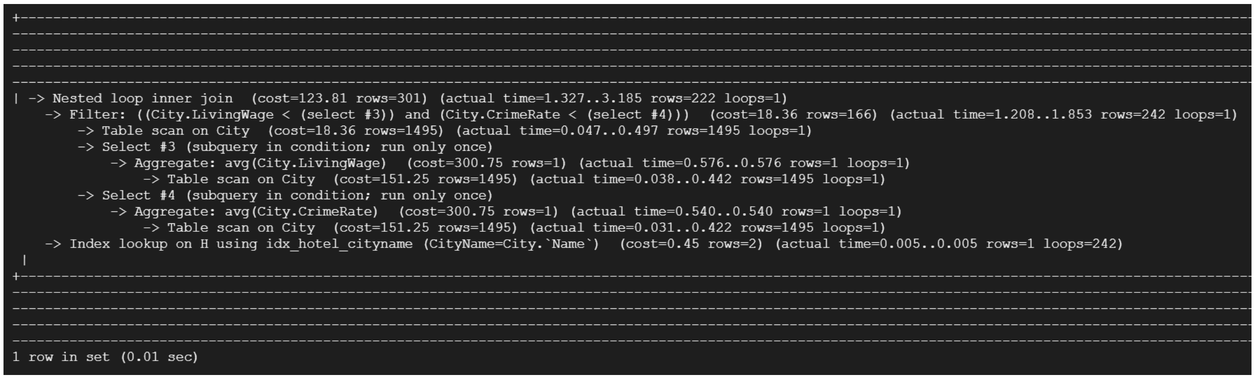
```
+----------------------------------------------------------+------------+-------------------------+--------------------+--------------------+
| name                                                     | cityName   | Address                 | LivingWage         | CrimeRate          |
+----------------------------------------------------------+------------+-------------------------+--------------------+--------------------+
| Best Western Plus-prairie Inn                            | Albany     | 1100 Price Rd Se        | 26.484166666666667 | 1332.7999892234802 |
| Ramada Plaza Albany                                      | Albany     | 3 Watervliet Ave. Ext.  | 26.484166666666667 | 1332.7999892234802 |
| Holiday Inn Express Alpharetta - Roswell                 | Alpharetta | 2950 Mansell Rd         | 26.484166666666667 |  399.6000053882599 |
| Wingate By Wyndham Alpharetta                            | Alpharetta | 1005 Kingswood Pl       | 26.484166666666667 |  399.6000053882599 |
| Ramada-ankeny                                            | Ankeny     | 133 Se Delaware Ave     | 26.271666666666665 | 2917.2999782562256 |
| Best Western Appleton Inn                                | Appleton   | 3033 W College Ave      | 26.837499999999995 |  964.8999977111816 |
| Copperleaf Hotel                                         | Appleton   | 300 W College Ave       | 26.837499999999995 |  964.8999977111816 |
| Fairfield Inn Appleton                                   | Appleton   | 132 N Mall Dr           | 26.837499999999995 |  964.8999977111816 |
| Comfort Suites Outlet Center                             | Asheville  | 890 Brevard Rd          | 26.797499999999996 |  2264.600009918213 |
| Americas Best Value Inn                                  | Auburn     | 170 Center St           | 24.356666666666666 | 1507.1000146865845 |
| Auburn Travelodge Inn & Suites                           | Auburn     | 9 16th St Nw            | 24.356666666666666 | 1507.1000146865845 |
| Hearthside Village Cottage Motel                         | Bethlehem  | 1267 Main St            | 24.617499999999996 |  1244.600020647049 |
| Historic Hotel Bethlehem                                 | Bethlehem  | 437 Main St             | 24.617499999999996 |  1244.600020647049 |
| Holiday Inn Express & Suites Bethlehem                   | Bethlehem  | 2201 Cherry Ln          | 24.617499999999996 |  1244.600020647049 |
| Residence Inn Allentown Bethlehem/Lehigh Valley Airport  | Bethlehem  | 2180 Motel Dr           | 24.617499999999996 |  1244.600020647049 |
+----------------------------------------------------------+------------+-------------------------+--------------------+--------------------+
15 rows in set (0.00 sec)
```

Command:



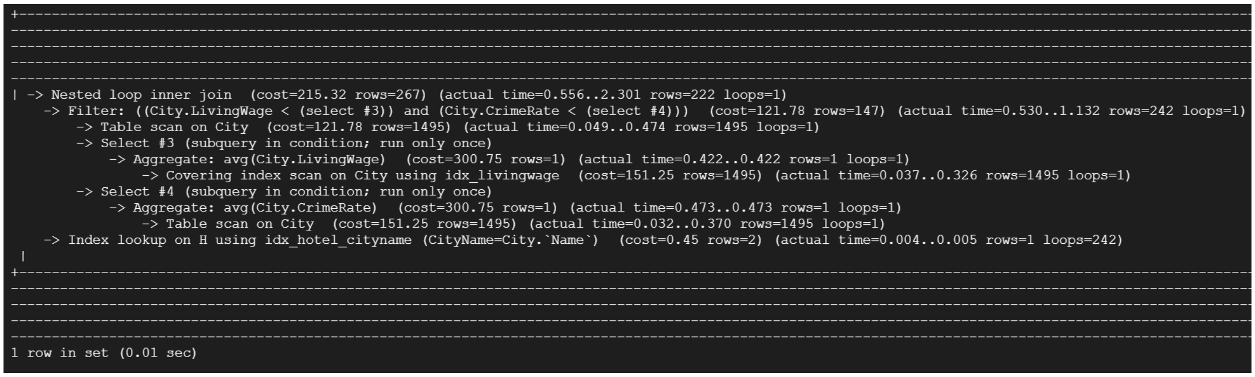
```
mysql> EXPLAIN ANALYZE
    -> SELECT H.name, H.cityName, H.Address, C.LivingWage, C.CrimeRate
    -> From Hotel H JOIN
    -> (SELECT * FROM City WHERE LivingWage < (SELECT AVG(LivingWage) FROM City) AND
    -> CrimeRate < (SELECT AVG(CrimeRate) FROM City)) C
    -> ON H.CityName = C.Name;
+----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
| EXPLAIN
```

Analysis before indexing:



```
+----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=123.81 rows=301) (actual time=1.327..3.185 rows=222 loops=1)
    -> Filter: ((City.LivingWage < (select #3)) and (City.CrimeRate < (select #4)))  (cost=18.36 rows=166) (actual time=1.208..1.853 rows=242 loops=1)
        -> Table scan on City  (cost=18.36 rows=1495) (actual time=0.047..0.497 rows=1495 loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(City.LivingWage)  (cost=300.75 rows=1) (actual time=0.576..0.576 rows=1 loops=1)
                -> Table scan on City  (cost=151.25 rows=1495) (actual time=0.038..0.442 rows=1495 loops=1)
        -> Select #4 (subquery in condition; run only once)
            -> Aggregate: avg(City.CrimeRate)  (cost=300.75 rows=1) (actual time=0.540..0.540 rows=1 loops=1)
                -> Table scan on City  (cost=151.25 rows=1495) (actual time=0.031..0.422 rows=1495 loops=1)
    -> Index lookup on H using idx_hotel_cityname (CityName=City.`Name`)  (cost=0.45 rows=2) (actual time=0.005..0.005 rows=1 loops=242)
 |
+----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

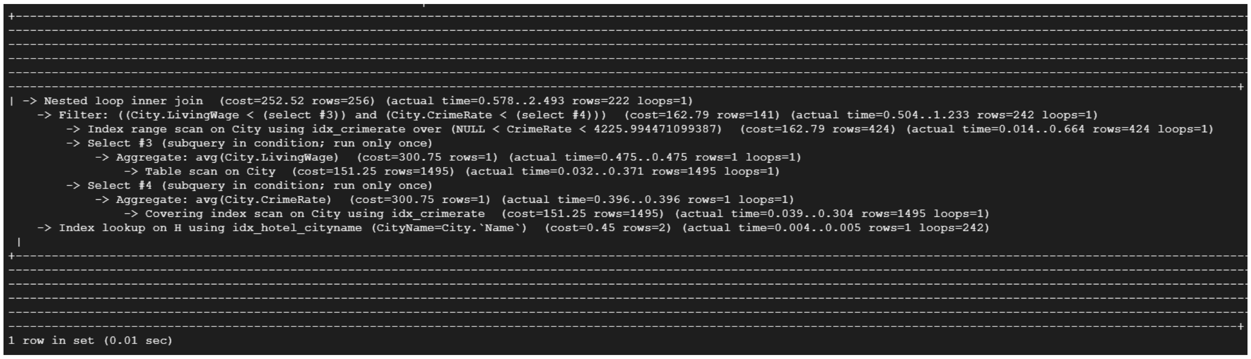
Attempt 1: Index on LivingWage



```
+----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
| -> Nested loop inner join  (cost=215.32 rows=267) (actual time=0.556..2.301 rows=222 loops=1)
    -> Filter: ((City.LivingWage < (select #3)) and (City.CrimeRate < (select #4)))  (cost=121.78 rows=147) (actual time=0.530..1.132 rows=242 loops=1)
        -> Table scan on City  (cost=121.78 rows=1495) (actual time=0.049..0.474 rows=1495 loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(City.LivingWage)  (cost=300.75 rows=1) (actual time=0.422..0.422 rows=1 loops=1)
                -> Covering index scan on City using idx_livingwage  (cost=151.25 rows=1495) (actual time=0.037..0.326 rows=1495 loops=1)
        -> Select #4 (subquery in condition; run only once)
            -> Aggregate: avg(City.CrimeRate)  (cost=300.75 rows=1) (actual time=0.473..0.473 rows=1 loops=1)
                -> Table scan on City  (cost=151.25 rows=1495) (actual time=0.032..0.370 rows=1495 loops=1)
    -> Index lookup on H using idx_hotel_cityname (CityName=City.`Name`)  (cost=0.45 rows=2) (actual time=0.004..0.005 rows=1 loops=242)
 |
+----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

Similarly, we expected to improve performance by indexing attributes in the WHERE clause, such as 'LivingWage'. Contrary to our expectations, not only did we fail to see a decrease in query cost, but we also experienced a decline in performance. We suspect that the relatively small size of our table is insufficient to demonstrate notable benefits. Furthermore, the additional overhead associated with implementing the index has negatively impacted overall performance.

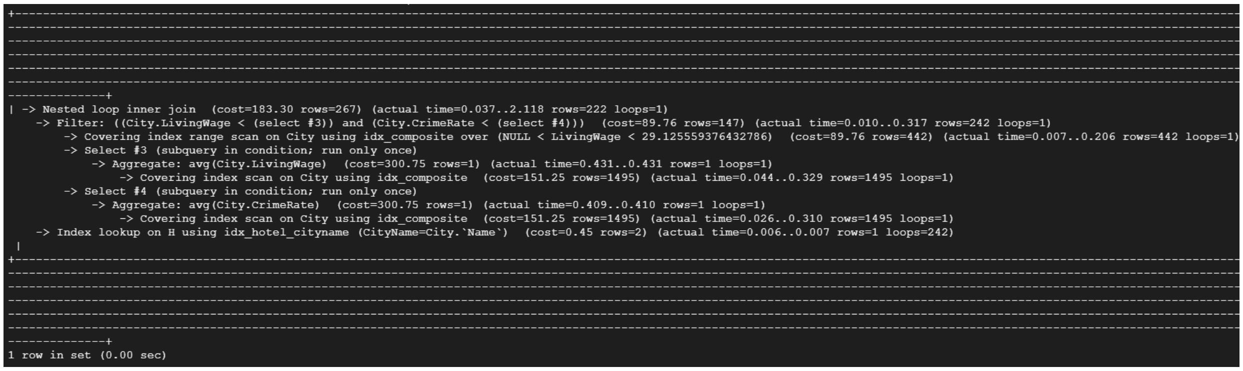
Attempt 2: Index on CrimeRate



```
+-------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
| -> Nested loop inner join  (cost=252.52 rows=256) (actual time=0.578..2.493 rows=222 loops=1)
    -> Filter: ((City.LivingWage < (select #3)) and (City.CrimeRate < (select #4)))  (cost=162.79 rows=141) (actual time=0.504..1.233 rows=242 loops=1)
        -> Index range scan on City using idx_crimerate over (NULL < CrimeRate < 4225.994471099387)  (cost=162.79 rows=424) (actual time=0.014..0.664 rows=424 loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(City.LivingWage)  (cost=300.75 rows=1) (actual time=0.475..0.475 rows=1 loops=1)
                -> Table scan on City  (cost=151.25 rows=1495) (actual time=0.032..0.371 rows=1495 loops=1)
        -> Select #4 (subquery in condition; run only once)
            -> Aggregate: avg(City.CrimeRate)  (cost=300.75 rows=1) (actual time=0.396..0.396 rows=1 loops=1)
                -> Covering index scan on City using idx_crimerate  (cost=151.25 rows=1495) (actual time=0.039..0.304 rows=1495 loops=1)
    -> Index lookup on H using idx_hotel_cityname (CityName=City.`Name`)  (cost=0.45 rows=2) (actual time=0.004..0.005 rows=1 loops=242)
 |
+-------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

Once again, we expected to improve performance by indexing attributes in the WHERE clause, such as 'CrimeRate'. Contrary to our expectations, not only did we fail to see a decrease in query cost, but we also experienced a decline in performance. We suspect that the relatively small size of our table is insufficient to demonstrate notable benefits. Furthermore, the additional overhead associated with implementing the index has negatively impacted overall performance.

Attempt 3: Index on LivingWage and CrimeRate (composite index)



```
+-------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------+
| -> Nested loop inner join  (cost=183.30 rows=267) (actual time=0.037..2.118 rows=222 loops=1)
    -> Filter: ((City.LivingWage < (select #3)) and (City.CrimeRate < (select #4)))  (cost=89.76 rows=147) (actual time=0.010..0.317 rows=242 loops=1)
        -> Covering index range scan on City using idx_composite over (NULL < LivingWage < 29.125559376432786)  (cost=89.76 rows=442) (actual time=0.007..0.206 rows=442 loops=1)
        -> Select #3 (subquery in condition; run only once)
            -> Aggregate: avg(City.LivingWage)  (cost=300.75 rows=1) (actual time=0.431..0.431 rows=1 loops=1)
                -> Covering index scan on City using idx_composite  (cost=151.25 rows=1495) (actual time=0.044..0.329 rows=1495 loops=1)
        -> Select #4 (subquery in condition; run only once)
            -> Aggregate: avg(City.CrimeRate)  (cost=300.75 rows=1) (actual time=0.409..0.410 rows=1 loops=1)
                -> Covering index scan on City using idx_composite  (cost=151.25 rows=1495) (actual time=0.026..0.310 rows=1495 loops=1)
    -> Index lookup on H using idx_hotel_cityname (CityName=City.`Name`)  (cost=0.45 rows=2) (actual time=0.006..0.007 rows=1 loops=242)
 |
+-------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------
-------------+
1 row in set (0.00 sec)
```

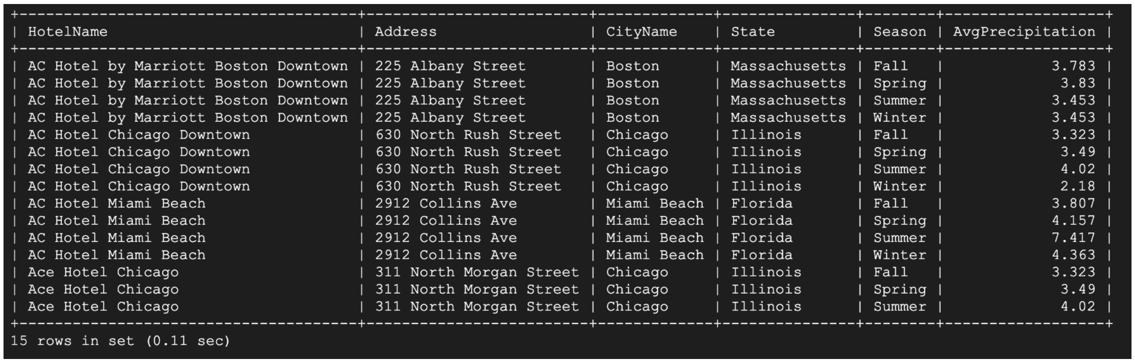
Creating a composite index on 'LivingWage' and ‘CrimeRate' has yielded the same results as previously observed.

Advanced Query 4

Advanced features: join multiple relations, subqueries

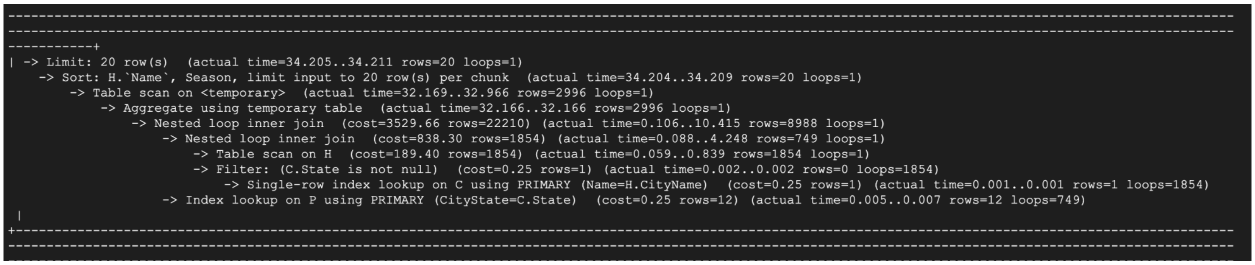
```
SELECT
        H.Name AS HotelName,
        H.Address,
        H.CityName,
        C.State,
CASE
        WHEN P.Month IN (3, 4, 5) THEN 'Spring'
        WHEN P.Month IN (6, 7, 8) THEN 'Summer'
        WHEN P.Month IN (9, 10, 11) THEN 'Fall'
        ELSE 'Winter'
END AS Season,
          ROUND(AVG(P.Precipitation), 3) AS AvgPrecipitation
FROM Hotel H
        JOIN City C ON H.CityName = C.Name
        JOIN Precipitation P ON C.State = P.CityState
GROUP BY
        H.Name,
        H.Address,
        H.CityName,
        C.State,
        Season
ORDER BY
        H.Name, Season
LIMIT 15;
```

Top 15 results:



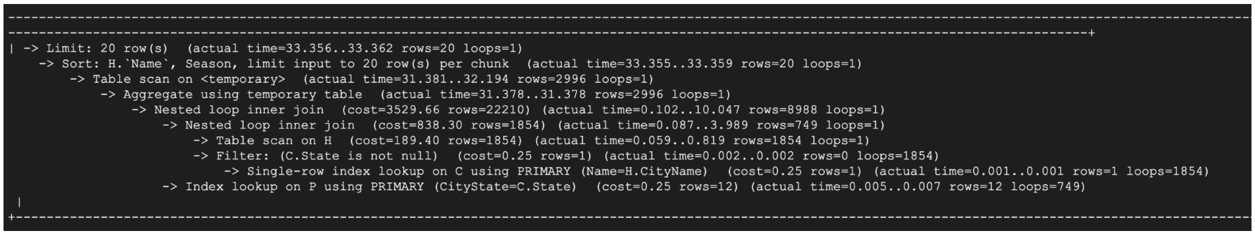
```
+--------------------------------------+-------------------------+-------------+---------------+--------+------------------+
| HotelName                            | Address                 | CityName    | State         | Season | AvgPrecipitation |
+--------------------------------------+-------------------------+-------------+---------------+--------+------------------+
| AC Hotel by Marriott Boston Downtown | 225 Albany Street       | Boston      | Massachusetts | Fall   |            3.783 |
| AC Hotel by Marriott Boston Downtown | 225 Albany Street       | Boston      | Massachusetts | Spring |             3.83 |
| AC Hotel by Marriott Boston Downtown | 225 Albany Street       | Boston      | Massachusetts | Summer |            3.453 |
| AC Hotel by Marriott Boston Downtown | 225 Albany Street       | Boston      | Massachusetts | Winter |            3.453 |
| AC Hotel Chicago Downtown            | 630 North Rush Street   | Chicago     | Illinois      | Fall   |            3.323 |
| AC Hotel Chicago Downtown            | 630 North Rush Street   | Chicago     | Illinois      | Spring |             3.49 |
| AC Hotel Chicago Downtown            | 630 North Rush Street   | Chicago     | Illinois      | Summer |             4.02 |
| AC Hotel Chicago Downtown            | 630 North Rush Street   | Chicago     | Illinois      | Winter |             2.18 |
| AC Hotel Miami Beach                 | 2912 Collins Ave        | Miami Beach | Florida       | Fall   |            3.807 |
| AC Hotel Miami Beach                 | 2912 Collins Ave        | Miami Beach | Florida       | Spring |            4.157 |
| AC Hotel Miami Beach                 | 2912 Collins Ave        | Miami Beach | Florida       | Summer |            7.417 |
| AC Hotel Miami Beach                 | 2912 Collins Ave        | Miami Beach | Florida       | Winter |            4.363 |
| Ace Hotel Chicago                    | 311 North Morgan Street | Chicago     | Illinois      | Fall   |            3.323 |
| Ace Hotel Chicago                    | 311 North Morgan Street | Chicago     | Illinois      | Spring |             3.49 |
| Ace Hotel Chicago                    | 311 North Morgan Street | Chicago     | Illinois      | Summer |             4.02 |
+--------------------------------------+-------------------------+-------------+---------------+--------+------------------+
15 rows in set (0.11 sec)
```

Analysis before indexing:



```
---------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
-----------+
| -> Limit: 20 row(s)  (actual time=34.205..34.211 rows=20 loops=1)
    -> Sort: H.`Name`, Season, limit input to 20 row(s) per chunk  (actual time=34.204..34.209 rows=20 loops=1)
        -> Table scan on <temporary>  (actual time=32.169..32.966 rows=2996 loops=1)
            -> Aggregate using temporary table  (actual time=32.166..32.166 rows=2996 loops=1)
                -> Nested loop inner join  (cost=3529.66 rows=22210) (actual time=0.106..10.415 rows=8988 loops=1)
                    -> Nested loop inner join  (cost=838.30 rows=1854) (actual time=0.088..4.248 rows=749 loops=1)
                        -> Table scan on H  (cost=189.40 rows=1854) (actual time=0.059..0.839 rows=1854 loops=1)
                        -> Filter: (C.State is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0 loops=1854)
                            -> Single-row index lookup on C using PRIMARY (Name=H.CityName)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1854)
                    -> Index lookup on P using PRIMARY (CityState=C.State)  (cost=0.25 rows=12) (actual time=0.005..0.007 rows=12 loops=749)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------------------
```

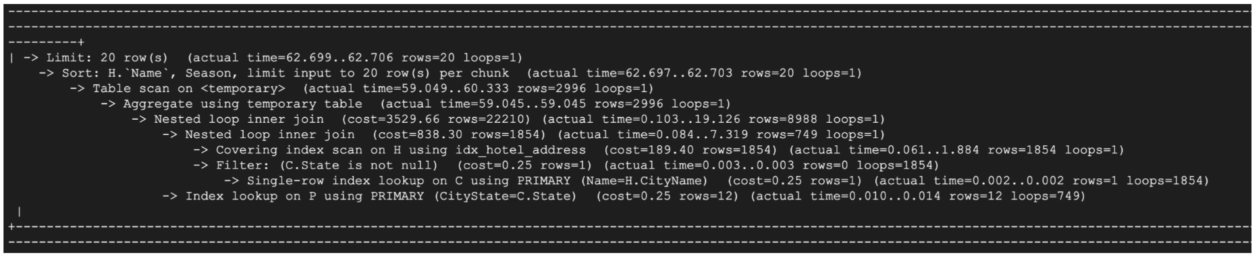
Attempt 1: Index on Precipitation



```
----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 20 row(s)  (actual time=33.356..33.362 rows=20 loops=1)
    -> Sort: H.`Name`, Season, limit input to 20 row(s) per chunk  (actual time=33.355..33.359 rows=20 loops=1)
        -> Table scan on <temporary>  (actual time=31.381..32.194 rows=2996 loops=1)
            -> Aggregate using temporary table  (actual time=31.378..31.378 rows=2996 loops=1)
                -> Nested loop inner join  (cost=3529.66 rows=22210) (actual time=0.102..10.047 rows=8988 loops=1)
                    -> Nested loop inner join  (cost=838.30 rows=1854) (actual time=0.087..3.989 rows=749 loops=1)
                        -> Table scan on H  (cost=189.40 rows=1854) (actual time=0.059..0.819 rows=1854 loops=1)
                        -> Filter: (C.State is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0 loops=1854)
                            -> Single-row index lookup on C using PRIMARY (Name=H.CityName)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1854)
                    -> Index lookup on P using PRIMARY (CityState=C.State)  (cost=0.25 rows=12) (actual time=0.005..0.007 rows=12 loops=749)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
```

The addition of an index on the attribute present in the SELECT clause, specifically 'Precipitation,' was not expected to influence the cost. The outcomes of our analysis align with these expectations, confirming that the index has no impact on query cost.

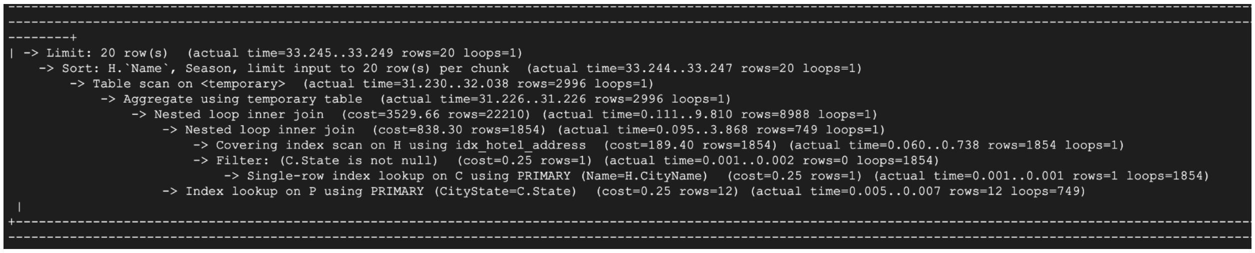
Attempt 2: Index on Address



```
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
---------+
| -> Limit: 20 row(s)  (actual time=62.699..62.706 rows=20 loops=1)
    -> Sort: H.`Name`, Season, limit input to 20 row(s) per chunk  (actual time=62.697..62.703 rows=20 loops=1)
        -> Table scan on <temporary>  (actual time=59.049..60.333 rows=2996 loops=1)
            -> Aggregate using temporary table  (actual time=59.045..59.045 rows=2996 loops=1)
                -> Nested loop inner join  (cost=3529.66 rows=22210) (actual time=0.103..19.126 rows=8988 loops=1)
                    -> Nested loop inner join  (cost=838.30 rows=1854) (actual time=0.084..7.319 rows=749 loops=1)
                        -> Covering index scan on H using idx_hotel_address  (cost=189.40 rows=1854) (actual time=0.061..1.884 rows=1854 loops=1)
                        -> Filter: (C.State is not null)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=0 loops=1854)
                            -> Single-row index lookup on C using PRIMARY (Name=H.CityName)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1854)
                    -> Index lookup on P using PRIMARY (CityState=C.State)  (cost=0.25 rows=12) (actual time=0.010..0.014 rows=12 loops=749)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

We aimed to boost performance by indexing attributes in the GROUP BY clause, like 'Address'. The idea was that indexing would quickly group identical values, enhancing efficiency. Yet, we haven't seen a decrease in query costs. It seems the scale of our table might not be large enough to manifest significant improvements, given that the advantages of indexing are counterbalanced by the implementation's overhead.

Attempt 3: Index on 'Precipitation' and 'Address'



```
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
--------+
| -> Limit: 20 row(s)  (actual time=33.245..33.249 rows=20 loops=1)
    -> Sort: H.`Name`, Season, limit input to 20 row(s) per chunk  (actual time=33.244..33.247 rows=20 loops=1)
        -> Table scan on <temporary>  (actual time=31.230..32.038 rows=2996 loops=1)
            -> Aggregate using temporary table  (actual time=31.226..31.226 rows=2996 loops=1)
                -> Nested loop inner join  (cost=3529.66 rows=22210) (actual time=0.111..9.810 rows=8988 loops=1)
                    -> Nested loop inner join  (cost=838.30 rows=1854) (actual time=0.095..3.868 rows=749 loops=1)
                        -> Covering index scan on H using idx_hotel_address  (cost=189.40 rows=1854) (actual time=0.060..0.738 rows=1854 loops=1)
                        -> Filter: (C.State is not null)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=0 loops=1854)
                            -> Single-row index lookup on C using PRIMARY (Name=H.CityName)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1854)
                    -> Index lookup on P using PRIMARY (CityState=C.State)  (cost=0.25 rows=12) (actual time=0.005..0.007 rows=12 loops=749)
 |
+---------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

Creating a composite index on 'Precipitation' and 'Address' has yielded the same results as previously observed.