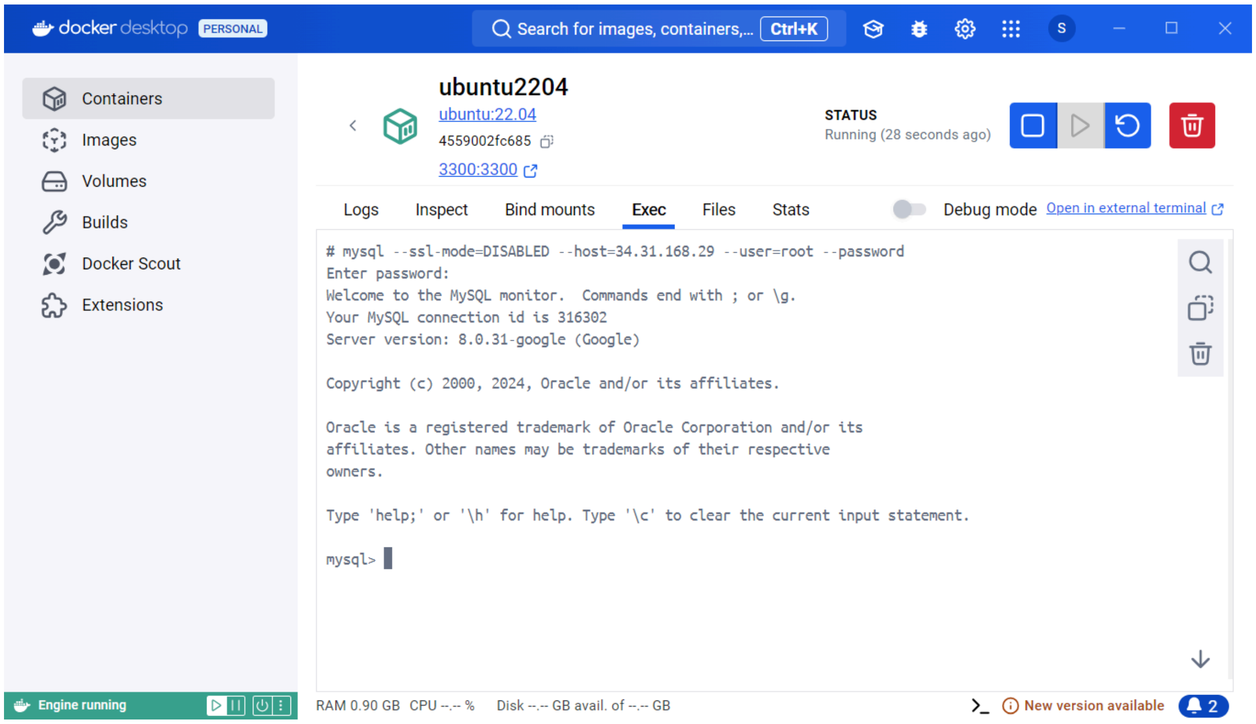
1. Database implementation

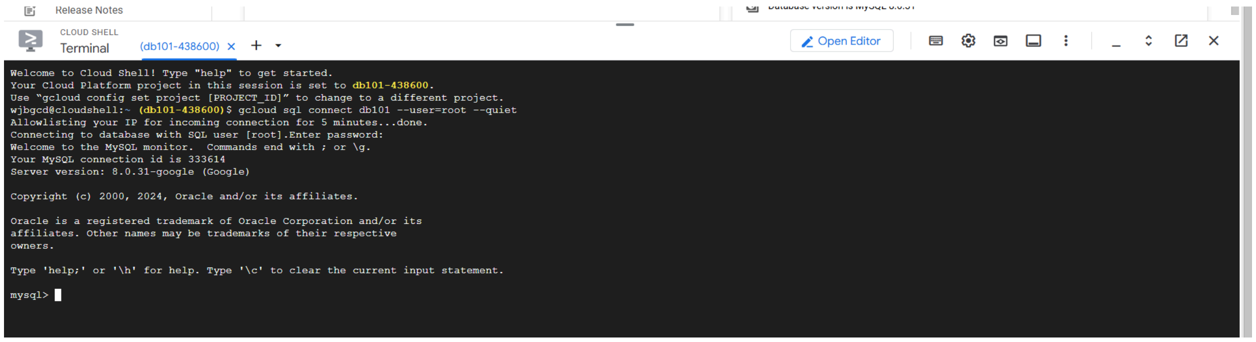
We built the MySQL database on GCP, below are the connection command and the cloud terminal:

Connection command :



```
# mysql --ssl-mode=DISABLED --host=34.31.168.29 --user=root --password
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 316302
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> 
```

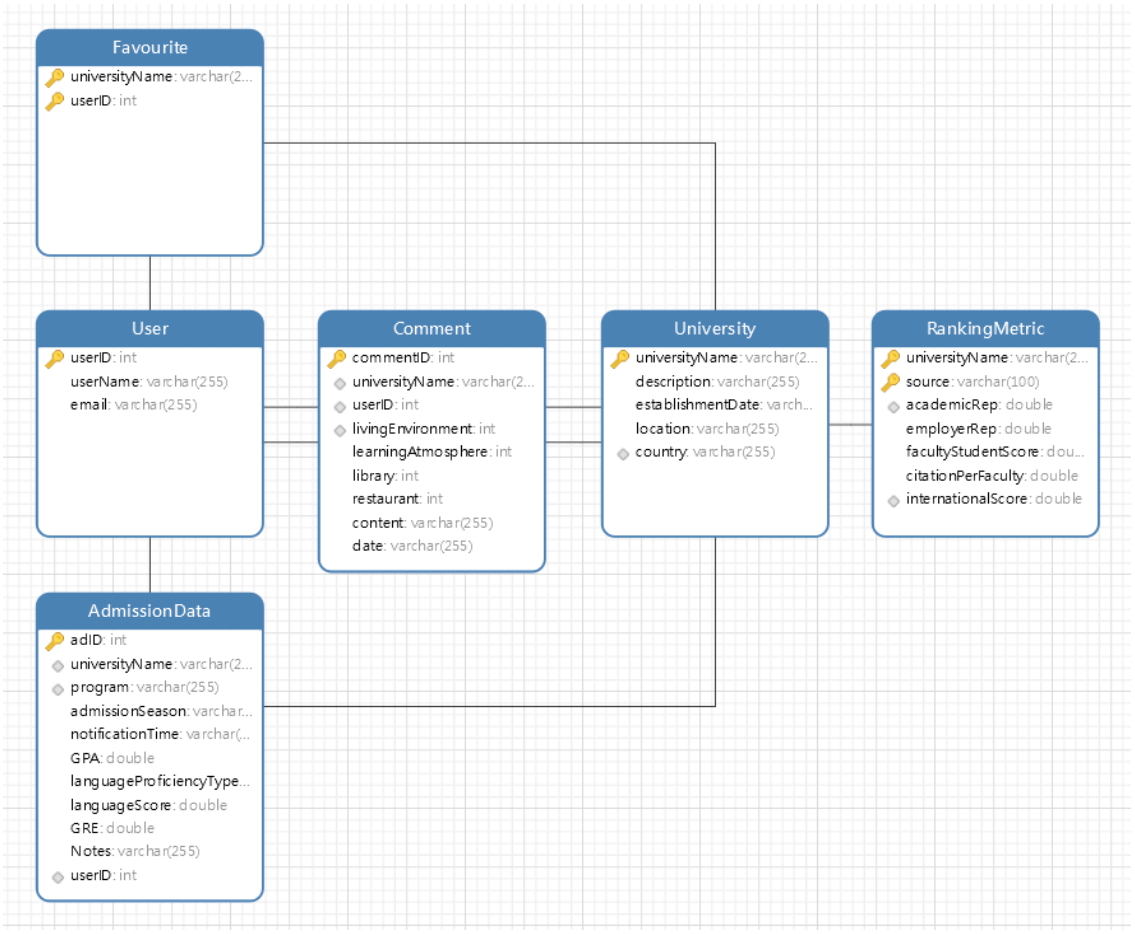
Cloud Shell:



```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to db101-438600.
Use "gcloud config set project [PROJECT_ID]" to change to a different project.
wjbgcd@cloudshell:~ (db101-438600)$ gcloud sql connect db101 --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 333614
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> 
```

We have six tables in total, the E-R diagram and the DDL are shown below.

E-R diagram:

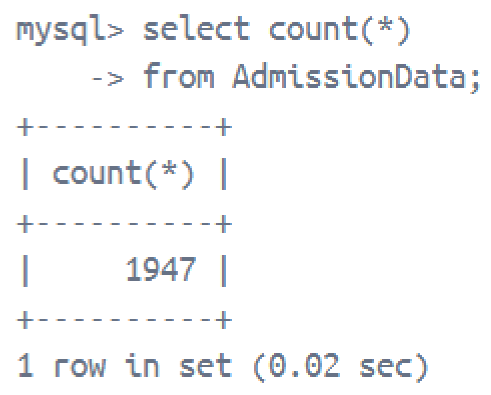


DDL:

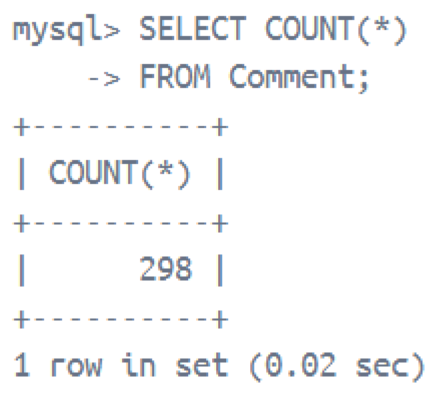
```sql
-- University Table
CREATE TABLE University (
    universityName VARCHAR(255) PRIMARY KEY,
    description VARCHAR(255),
    establishmentDate VARCHAR(255),
    location VARCHAR(255),
    country VARCHAR(255)
);

-- RankingMetric Table
CREATE TABLE RankingMetric (
    universityName VARCHAR(255),
    source VARCHAR(100),
    academicRep DOUBLE,
    employerRep DOUBLE,
    facultyStudentScore DOUBLE,
    citationPerFaculty DOUBLE,
    internationalScore DOUBLE,
    PRIMARY KEY (universityName, source),
    FOREIGN KEY (universityName) REFERENCES University(universityName)
);

-- User Table
CREATE TABLE User (
    userID INT PRIMARY KEY,
    userName VARCHAR(255),
    email VARCHAR(255)
);

-- Favourite Table
CREATE TABLE Favourite (
    universityName VARCHAR(255),
    userID INT,
    PRIMARY KEY (universityName, userID),
    FOREIGN KEY (universityName) REFERENCES University(universityName),
    FOREIGN KEY (userID) REFERENCES User(userID)
);

-- Comment Table
CREATE TABLE Comment (
    commentID INT PRIMARY KEY,
    universityName VARCHAR(255),
    userID INT,
    livingEnvironment INT,
    learningAtmosphere INT,
    library INT,
    restaurant INT,
    content VARCHAR(255),
    date VARCHAR(255),
    FOREIGN KEY (universityName) REFERENCES University(universityName),
    FOREIGN KEY (userID) REFERENCES User(userID)
);

-- AdmissionData Table
```

```
CREATE TABLE AdmissionData (
   adID INT PRIMARY KEY,
   userID INT,
   universityName VARCHAR(255),
   program VARCHAR(255),
   admissionSeason VARCHAR(255),
   notificationTime VARCHAR(255),
   GPA DOUBLE,
   languageProficiencyType VARCHAR(255),
   languageScore DOUBLE,
   GRE DOUBLE,
   Notes VARCHAR(255),
   FOREIGN KEY (userID) REFERENCES User(userID),
   FOREIGN KEY (universityName) REFERENCES University(universityName)
);
```

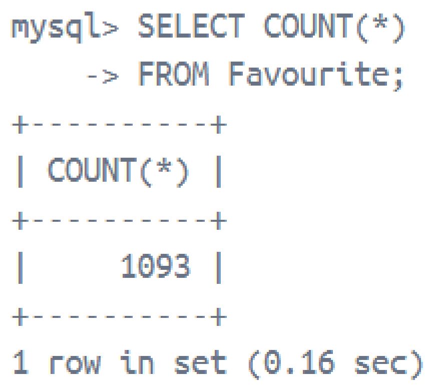
Data insertion:
After creating the tables, we inserted data into the tables. We inserted generated data into the Comment, Favourite, and User tables, and real data into the AdmissionData, RankingMetric, and University tables. To meet the requirements of stage 3, we inserted over 1000 records into the AdmissionData, Favourite, and RankingMetric tables.



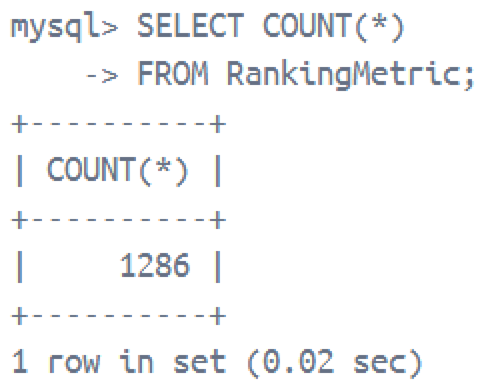
```
mysql> select count(*)
    -> from AdmissionData;
+----------+
| count(*) |
+----------+
|     1947 |
+----------+
1 row in set (0.02 sec)
```



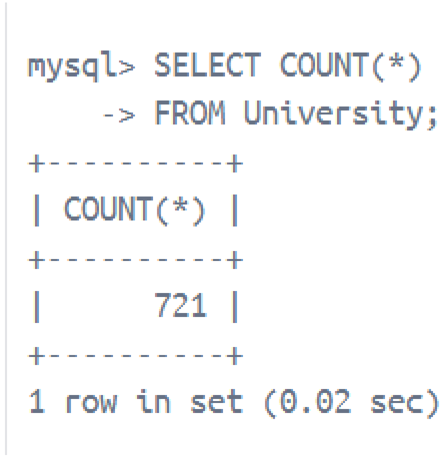
```
mysql> SELECT COUNT(*)
    -> FROM Comment;
+----------+
| COUNT(*) |
+----------+
|      298 |
+----------+
1 row in set (0.02 sec)
```



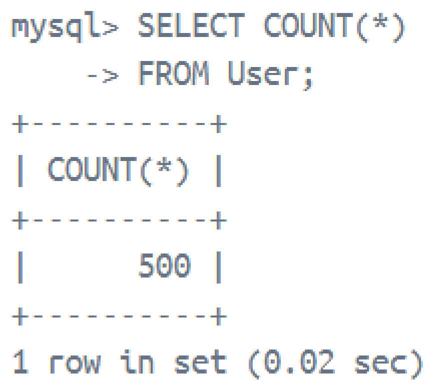
```
mysql> SELECT COUNT(*)
    -> FROM Favourite;
+----------+
| COUNT(*) |
+----------+
|     1093 |
+----------+
1 row in set (0.16 sec)
```



```
mysql> SELECT COUNT(*)
    -> FROM RankingMetric;
+----------+
| COUNT(*) |
+----------+
|     1286 |
+----------+
1 row in set (0.02 sec)
```



```
mysql> SELECT COUNT(*)
    -> FROM University;
+----------+
| COUNT(*) |
+----------+
|      721 |
+----------+
1 row in set (0.02 sec)
```



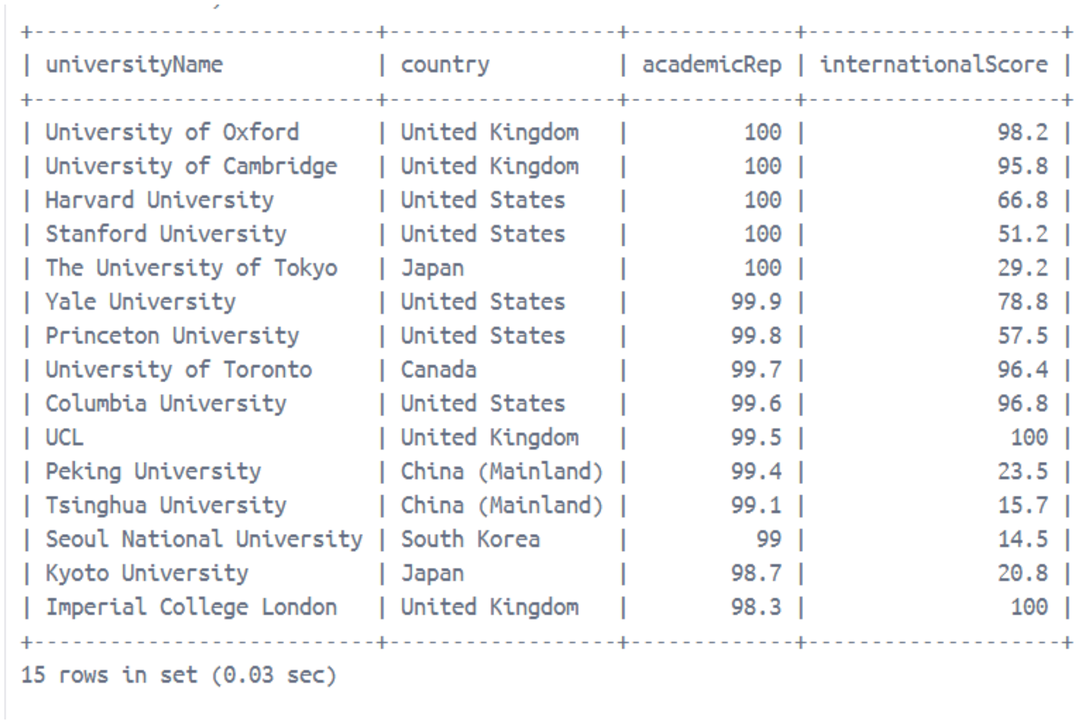
```
mysql> SELECT COUNT(*)
    -> FROM User;
+----------+
| COUNT(*) |
+----------+
|      500 |
+----------+
1 row in set (0.02 sec)
```

2. Advanced queries and according index design

- Rank schools based on different metrics (taking the example of using QS data and querying for academic reputation or international score greater than the average)

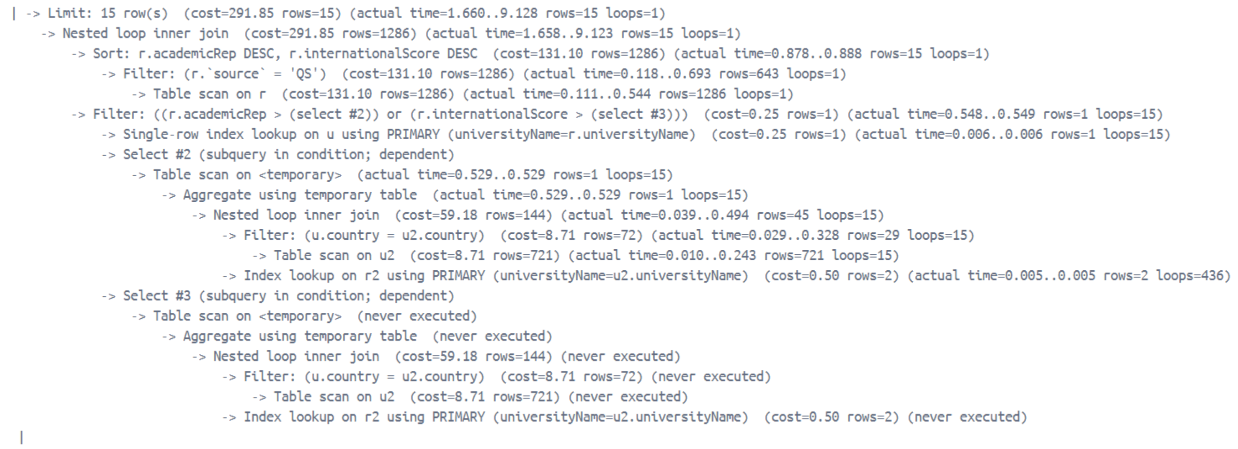
```
SELECT u.universityName,u.country,r.academicRep,r.internationalScore
FROM RankingMetric r JOIN University u on r.universityName = u.universityName
WHERE r.source='QS' AND ( r.academicRep > (
                                    SELECT AVG(r2.academicRep)
                                    FROM RankingMetric r2 JOIN University u2 on r2.universityName =
u2.universityName
                                    WHERE u.country = u2.country
                                    GROUP BY u2.country
                                  )
                OR r.internationalScore > (
                                    SELECT AVG(r2.internationalScore)
                                    FROM RankingMetric r2 JOIN University u2 on r2.universityName =
u2.universityName
                                    WHERE u.country = u2.country
```

```
                                GROUP BY u2.country
                            )
                    )
ORDER BY r.academicRep DESC, r.internationalScore DESC
LIMIT 15;
```

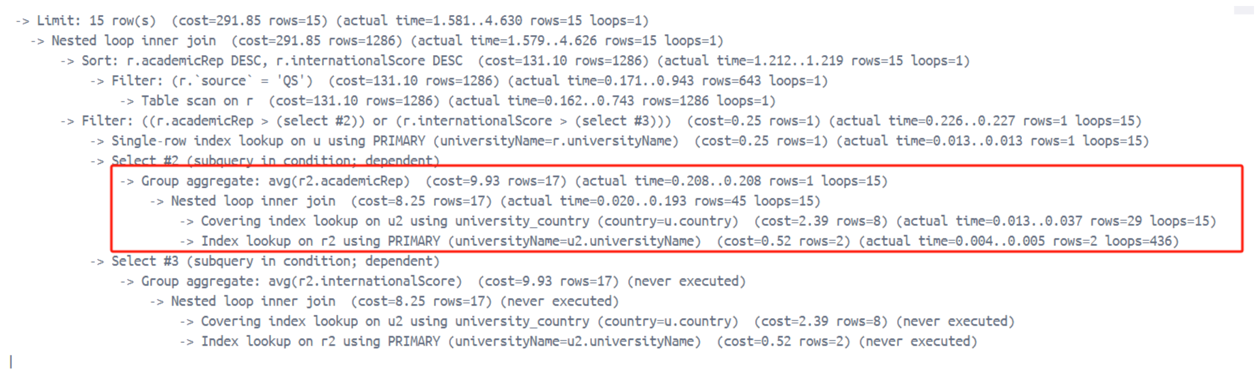


```
+--------------------------+-----------------+-------------+--------------------+
| universityName           | country         | academicRep | internationalScore |
+--------------------------+-----------------+-------------+--------------------+
| University of Oxford     | United Kingdom  |         100 |               98.2 |
| University of Cambridge  | United Kingdom  |         100 |               95.8 |
| Harvard University       | United States   |         100 |               66.8 |
| Stanford University      | United States   |         100 |               51.2 |
| The University of Tokyo  | Japan           |         100 |               29.2 |
| Yale University          | United States   |        99.9 |               78.8 |
| Princeton University     | United States   |        99.8 |               57.5 |
| University of Toronto    | Canada          |        99.7 |               96.4 |
| Columbia University      | United States   |        99.6 |               96.8 |
| UCL                      | United Kingdom  |        99.5 |                100 |
| Peking University        | China (Mainland) |       99.4 |               23.5 |
| Tsinghua University      | China (Mainland) |       99.1 |               15.7 |
| Seoul National University | South Korea    |          99 |               14.5 |
| Kyoto University         | Japan           |        98.7 |               20.8 |
| Imperial College London  | United Kingdom  |        98.3 |                100 |
+--------------------------+-----------------+-------------+--------------------+
15 rows in set (0.03 sec)
```

The original analysis is shown below.

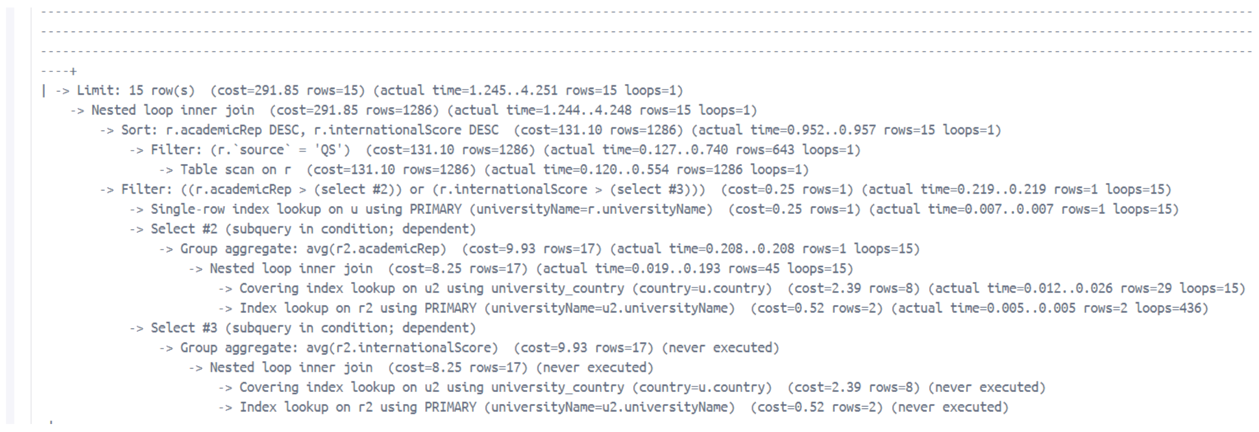


```
| -> Limit: 15 row(s)  (cost=291.85 rows=15) (actual time=1.660..9.128 rows=15 loops=1)
    -> Nested loop inner join  (cost=291.85 rows=1286) (actual time=1.658..9.123 rows=15 loops=1)
        -> Sort: r.academicRep DESC, r.internationalScore DESC  (cost=131.10 rows=1286) (actual time=0.878..0.888 rows=15 loops=1)
            -> Filter: (r.`source` = 'QS')  (cost=131.10 rows=1286) (actual time=0.118..0.693 rows=643 loops=1)
                -> Table scan on r  (cost=131.10 rows=1286) (actual time=0.111..0.544 rows=1286 loops=1)
        -> Filter: ((r.academicRep > (select #2)) or (r.internationalScore > (select #3)))  (cost=0.25 rows=1) (actual time=0.548..0.549 rows=1 loops=15)
            -> Single-row index lookup on u using PRIMARY (universityName=r.universityName)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=15)
            -> Select #2 (subquery in condition; dependent)
                -> Table scan on <temporary>  (actual time=0.529..0.529 rows=1 loops=15)
                    -> Aggregate using temporary table  (actual time=0.529..0.529 rows=1 loops=15)
                        -> Nested loop inner join  (cost=59.18 rows=144) (actual time=0.039..0.494 rows=45 loops=15)
                            -> Filter: (u.country = u2.country)  (cost=8.71 rows=72) (actual time=0.029..0.328 rows=29 loops=15)
                                -> Table scan on u2  (cost=8.71 rows=721) (actual time=0.010..0.243 rows=721 loops=15)
                            -> Index lookup on r2 using PRIMARY (universityName=u2.universityName)  (cost=0.50 rows=2) (actual time=0.005..0.005 rows=2 loops=436)
            -> Select #3 (subquery in condition; dependent)
                -> Table scan on <temporary>  (never executed)
                    -> Aggregate using temporary table  (never executed)
                        -> Nested loop inner join  (cost=59.18 rows=144) (never executed)
                            -> Filter: (u.country = u2.country)  (cost=8.71 rows=72) (never executed)
                                -> Table scan on u2  (cost=8.71 rows=721) (never executed)
                            -> Index lookup on r2 using PRIMARY (universityName=u2.universityName)  (cost=0.50 rows=2) (never executed)
 |
```



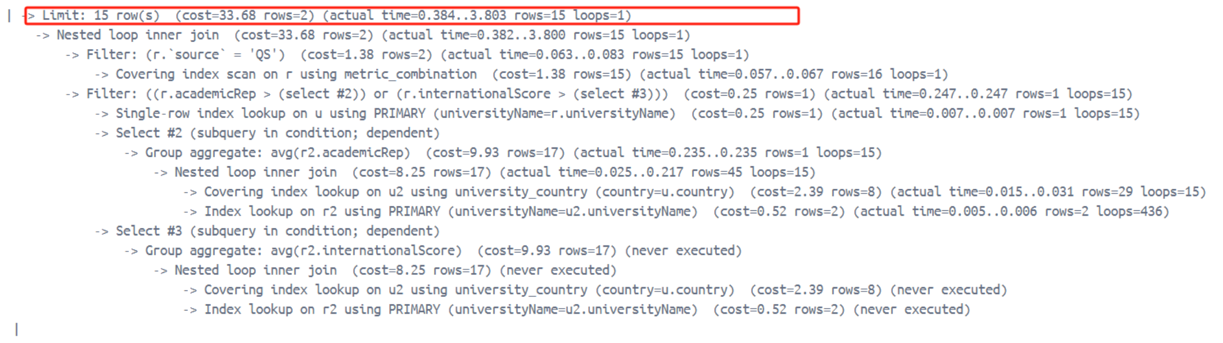
```
-> Limit: 15 row(s)  (cost=291.85 rows=15) (actual time=1.581..4.630 rows=15 loops=1)
  -> Nested loop inner join  (cost=291.85 rows=1286) (actual time=1.579..4.626 rows=15 loops=1)
      -> Sort: r.academicRep DESC, r.internationalScore DESC  (cost=131.10 rows=1286) (actual time=1.212..1.219 rows=15 loops=1)
          -> Filter: (r.`source` = 'QS')  (cost=131.10 rows=1286) (actual time=0.171..0.943 rows=643 loops=1)
              -> Table scan on r  (cost=131.10 rows=1286) (actual time=0.162..0.743 rows=1286 loops=1)
      -> Filter: ((r.academicRep > (select #2)) or (r.internationalScore > (select #3)))  (cost=0.25 rows=1) (actual time=0.226..0.227 rows=1 loops=15)
          -> Single-row index lookup on u using PRIMARY (universityName=r.universityName)  (cost=0.25 rows=1) (actual time=0.013..0.013 rows=1 loops=15)
          -> Select #2 (subquery in condition; dependent)
              -> Group aggregate: avg(r2.academicRep)  (cost=9.93 rows=17) (actual time=0.208..0.208 rows=1 loops=15)
                  -> Nested loop inner join  (cost=8.25 rows=17) (actual time=0.020..0.193 rows=45 loops=15)
                      -> Covering index lookup on u2 using university_country (country=u.country)  (cost=2.39 rows=8) (actual time=0.013..0.037 rows=29 loops=15)
                      -> Index lookup on r2 using PRIMARY (universityName=u2.universityName)  (cost=0.52 rows=2) (actual time=0.004..0.005 rows=2 loops=436)
          -> Select #3 (subquery in condition; dependent)
              -> Group aggregate: avg(r2.internationalScore)  (cost=9.93 rows=17) (never executed)
                  -> Nested loop inner join  (cost=8.25 rows=17) (never executed)
                      -> Covering index lookup on u2 using university_country (country=u.country)  (cost=2.39 rows=8) (never executed)
                      -> Index lookup on r2 using PRIMARY (universityName=u2.universityName)  (cost=0.52 rows=2) (never executed)
|
```

I firstly chose to add an index : university_country on University(country). This index works because the country index helps narrow the table scan to an index scan(from 721 row to 8 row)



```
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
----+
| -> Limit: 15 row(s)  (cost=291.85 rows=15) (actual time=1.245..4.251 rows=15 loops=1)
    -> Nested loop inner join  (cost=291.85 rows=1286) (actual time=1.244..4.248 rows=15 loops=1)
        -> Sort: r.academicRep DESC, r.internationalScore DESC  (cost=131.10 rows=1286) (actual time=0.952..0.957 rows=15 loops=1)
            -> Filter: (r.`source` = 'QS')  (cost=131.10 rows=1286) (actual time=0.127..0.740 rows=643 loops=1)
                -> Table scan on r  (cost=131.10 rows=1286) (actual time=0.120..0.554 rows=1286 loops=1)
        -> Filter: ((r.academicRep > (select #2)) or (r.internationalScore > (select #3)))  (cost=0.25 rows=1) (actual time=0.219..0.219 rows=1 loops=15)
            -> Single-row index lookup on u using PRIMARY (universityName=r.universityName)  (cost=0.25 rows=1) (actual time=0.007..0.007 rows=1 loops=15)
            -> Select #2 (subquery in condition; dependent)
                -> Group aggregate: avg(r2.academicRep)  (cost=9.93 rows=17) (actual time=0.208..0.208 rows=1 loops=15)
                    -> Nested loop inner join  (cost=8.25 rows=17) (actual time=0.019..0.193 rows=45 loops=15)
                        -> Covering index lookup on u2 using university_country (country=u.country)  (cost=2.39 rows=8) (actual time=0.012..0.026 rows=29 loops=15)
                        -> Index lookup on r2 using PRIMARY (universityName=u2.universityName)  (cost=0.52 rows=2) (actual time=0.005..0.005 rows=2 loops=436)
            -> Select #3 (subquery in condition; dependent)
                -> Group aggregate: avg(r2.internationalScore)  (cost=9.93 rows=17) (never executed)
                    -> Nested loop inner join  (cost=8.25 rows=17) (never executed)
                        -> Covering index lookup on u2 using university_country (country=u.country)  (cost=2.39 rows=8) (never executed)
                        -> Index lookup on r2 using PRIMARY (universityName=u2.universityName)  (cost=0.52 rows=2) (never executed)
```

I also tried to add an index on metric : academic_metric on RankingMetric(academicRep), but it didn't improve the efficiency. I think this is because this index is not that useful as the index that the PK has(universityName,source).



```
| -> Limit: 15 row(s)  (cost=33.68 rows=2) (actual time=0.384..3.803 rows=15 loops=1)
    -> Nested loop inner join  (cost=33.68 rows=2) (actual time=0.382..3.800 rows=15 loops=1)
        -> Filter: (r.`source` = 'QS')  (cost=1.38 rows=2) (actual time=0.063..0.083 rows=15 loops=1)
            -> Covering index scan on r using metric_combination  (cost=1.38 rows=15) (actual time=0.057..0.067 rows=16 loops=1)
        -> Filter: ((r.academicRep > (select #2)) or (r.internationalScore > (select #3)))  (cost=0.25 rows=1) (actual time=0.247..0.247 rows=1 loops=15)
            -> Single-row index lookup on u using PRIMARY (universityName=r.universityName)  (cost=0.25 rows=1) (actual time=0.007..0.007 rows=1 loops=15)
            -> Select #2 (subquery in condition; dependent)
                -> Group aggregate: avg(r2.academicRep)  (cost=9.93 rows=17) (actual time=0.235..0.235 rows=1 loops=15)
                    -> Nested loop inner join  (cost=8.25 rows=17) (actual time=0.025..0.217 rows=45 loops=15)
                        -> Covering index lookup on u2 using university_country (country=u.country)  (cost=2.39 rows=8) (actual time=0.015..0.031 rows=29 loops=15)
                        -> Index lookup on r2 using PRIMARY (universityName=u2.universityName)  (cost=0.52 rows=2) (actual time=0.005..0.006 rows=2 loops=436)
            -> Select #3 (subquery in condition; dependent)
                -> Group aggregate: avg(r2.internationalScore)  (cost=9.93 rows=17) (never executed)
                    -> Nested loop inner join  (cost=8.25 rows=17) (never executed)
                        -> Covering index lookup on u2 using university_country (country=u.country)  (cost=2.39 rows=8) (never executed)
                        -> Index lookup on r2 using PRIMARY (universityName=u2.universityName)  (cost=0.52 rows=2) (never executed)
 |
```

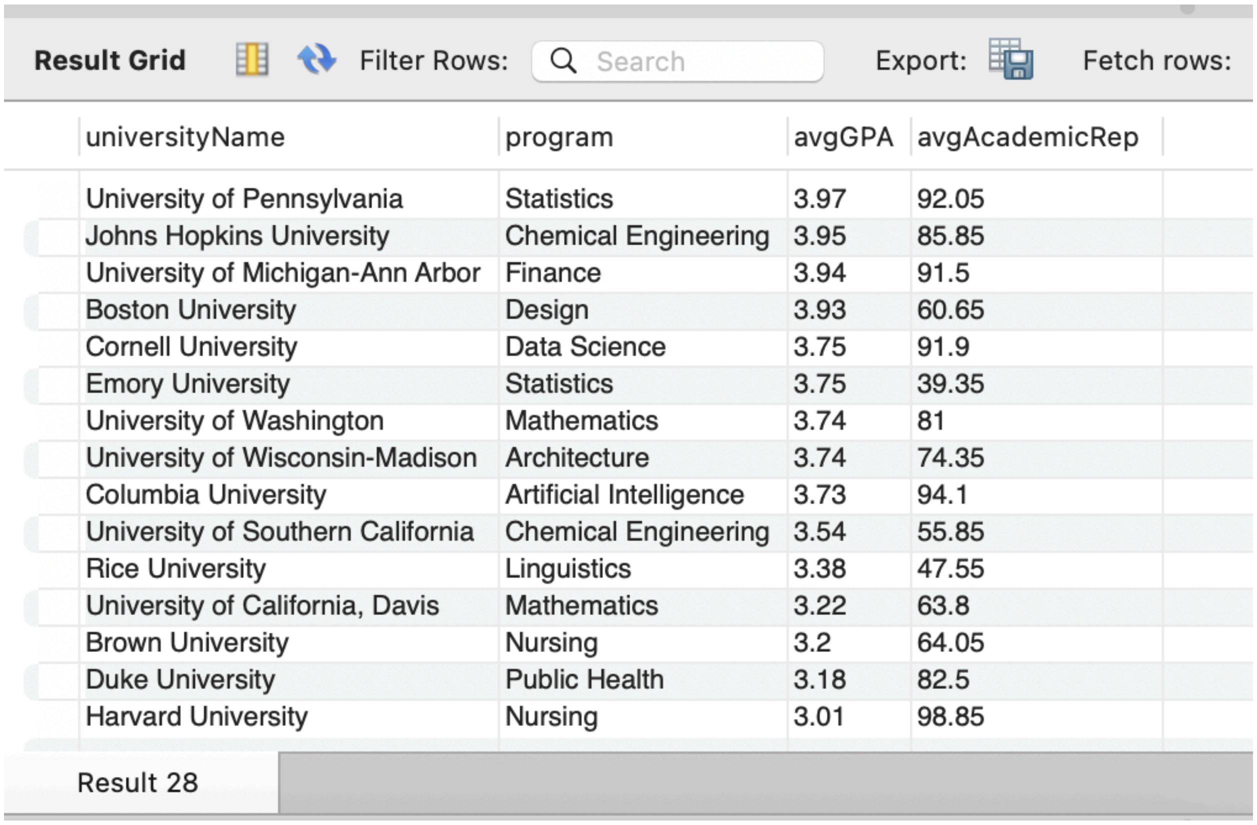
I also tried the composite Index to improve the efficiency(academicRep DESC, internationalScore DESC). It did help, because in that case it could get the query results in the descending order of the particular metrics, which will reduce the total amount of table scan if we use the limit. But I eventually decided not to use such indexes. The reason is that for this query's functionality, users have the flexibility to query any combination of metrics they choose.

Creating indexes for all possible metric combinations would incur an extremely high storage cost and maintenance overhead. So, it's not a good trade off.

In general, for this advanced query, we eventually just used the university_country index. It improves the efficiency of the subqueries, from cost = 77.1(before) to cost = 21.09(after).

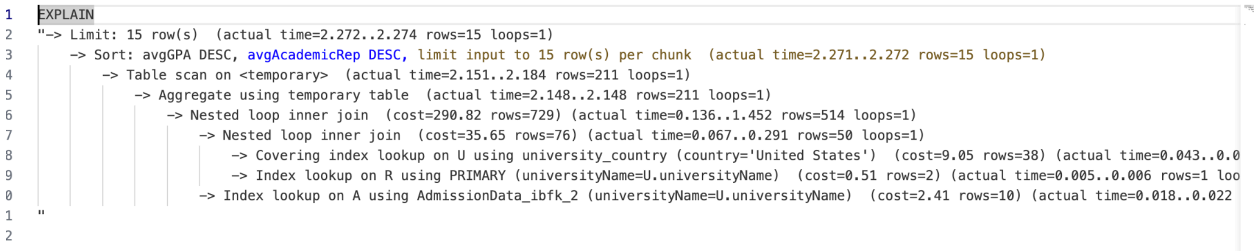
- Query for the average admission GPA and academic reputation scores for each program at U.S. universities

```
SELECT
   universityName,
   program,
   ROUND(AVG(averageGPA), 2) AS avgGPA,
   ROUND(AVG(academicRep), 2) AS avgAcademicRep
FROM
   (SELECT
      A.program,
      U.universityName,
      A.GPA AS averageGPA,
      R.academicRep
   FROM
      AdmissionData A
   JOIN
      University U ON A.universityName = U.universityName
   JOIN
      RankingMetric R ON U.universityName = R.universityName
   WHERE
      U.country = 'United States'
   ) AS clc
GROUP BY
   universityName, program
ORDER BY
   avgGPA DESC, avgAcademicRep DESC
LIMIT 15;
```



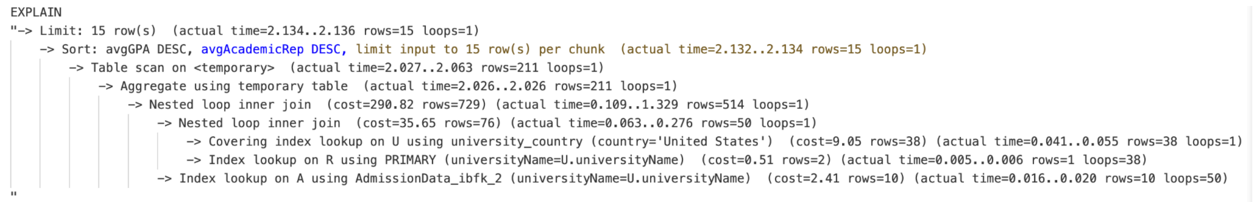
universityName	program	avgGPA	avgAcademicRep
University of Pennsylvania	Statistics	3.97	92.05
Johns Hopkins University	Chemical Engineering	3.95	85.85
University of Michigan-Ann Arbor	Finance	3.94	91.5
Boston University	Design	3.93	60.65
Cornell University	Data Science	3.75	91.9
Emory University	Statistics	3.75	39.35
University of Washington	Mathematics	3.74	81
University of Wisconsin-Madison	Architecture	3.74	74.35
Columbia University	Artificial Intelligence	3.73	94.1
University of Southern California	Chemical Engineering	3.54	55.85
Rice University	Linguistics	3.38	47.55
University of California, Davis	Mathematics	3.22	63.8
Brown University	Nursing	3.2	64.05
Duke University	Public Health	3.18	82.5
Harvard University	Nursing	3.01	98.85

Before adding index, the result is:



```
EXPLAIN
"-> Limit: 15 row(s)  (actual time=2.272..2.274 rows=15 loops=1)
    -> Sort: avgGPA DESC, avgAcademicRep DESC, limit input to 15 row(s) per chunk  (actual time=2.271..2.272 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.151..2.184 rows=211 loops=1)
            -> Aggregate using temporary table  (actual time=2.148..2.148 rows=211 loops=1)
                -> Nested loop inner join  (cost=290.82 rows=729) (actual time=0.136..1.452 rows=514 loops=1)
                    -> Nested loop inner join  (cost=35.65 rows=76) (actual time=0.067..0.291 rows=50 loops=1)
                        -> Covering index lookup on U using university_country (country='United States')  (cost=9.05 rows=38) (actual time=0.043..0.0
                        -> Index lookup on R using PRIMARY (universityName=U.universityName)  (cost=0.51 rows=2) (actual time=0.005..0.006 rows=1 loo
                    -> Index lookup on A using AdmissionData_ibfk_2 (universityName=U.universityName)  (cost=2.41 rows=10) (actual time=0.018..0.022
"
```

Test: idx_admission_program
(CREATE INDEX idx_admission_program ON AdmissionData(program);)

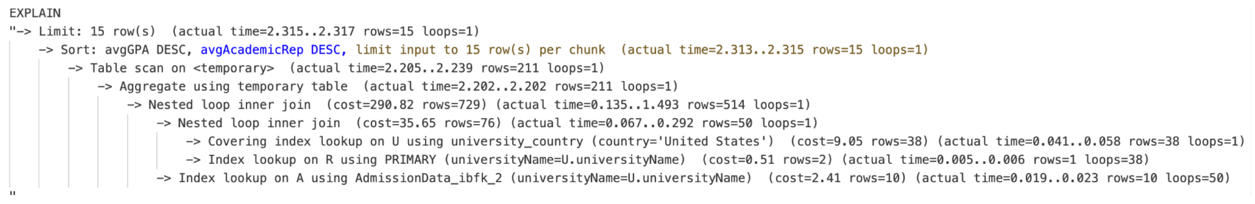


```
EXPLAIN
"-> Limit: 15 row(s)  (actual time=2.134..2.136 rows=15 loops=1)
    -> Sort: avgGPA DESC, avgAcademicRep DESC, limit input to 15 row(s) per chunk  (actual time=2.132..2.134 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.027..2.063 rows=211 loops=1)
            -> Aggregate using temporary table  (actual time=2.026..2.026 rows=211 loops=1)
                -> Nested loop inner join  (cost=290.82 rows=729) (actual time=0.109..1.329 rows=514 loops=1)
                    -> Nested loop inner join  (cost=35.65 rows=76) (actual time=0.063..0.276 rows=50 loops=1)
                        -> Covering index lookup on U using university_country (country='United States')  (cost=9.05 rows=38) (actual time=0.041..0.055 rows=38 loops=1)
                        -> Index lookup on R using PRIMARY (universityName=U.universityName)  (cost=0.51 rows=2) (actual time=0.005..0.006 rows=1 loops=38)
                    -> Index lookup on A using AdmissionData_ibfk_2 (universityName=U.universityName)  (cost=2.41 rows=10) (actual time=0.016..0.020 rows=10 loops=50)
"
```

As we can see, we added the index idx_admission_program. After testing, we observed a reduction in both time and cost in the nested loop inner join between this university (U) and the admission data (A) on the condition A.universityName = U.universityName.

Test Composite Index idx_universityname_rankingmetric and idx_rankingmetric_universityname
(CREATE INDEX idx_universityname_rankingmetric ON University(universityName);

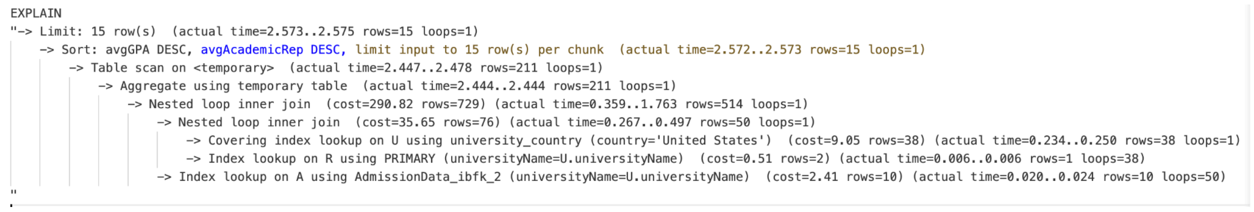
CREATE INDEX idx_rankingmetric_universityname ON RankingMetric(universityName);)



```
EXPLAIN
"-> Limit: 15 row(s)  (actual time=2.315..2.317 rows=15 loops=1)
    -> Sort: avgGPA DESC, avgAcademicRep DESC, limit input to 15 row(s) per chunk  (actual time=2.313..2.315 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.205..2.239 rows=211 loops=1)
            -> Aggregate using temporary table  (actual time=2.202..2.202 rows=211 loops=1)
                -> Nested loop inner join  (cost=290.82 rows=729) (actual time=0.135..1.493 rows=514 loops=1)
                    -> Nested loop inner join  (cost=35.65 rows=76) (actual time=0.067..0.292 rows=50 loops=1)
                        -> Covering index lookup on U using university_country (country='United States')  (cost=9.05 rows=38) (actual time=0.041..0.058 rows=38 loops=1)
                        -> Index lookup on R using PRIMARY (universityName=U.universityName)  (cost=0.51 rows=2) (actual time=0.005..0.006 rows=1 loops=38)
                    -> Index lookup on A using AdmissionData_ibfk_2 (universityName=U.universityName)  (cost=2.41 rows=10) (actual time=0.019..0.023 rows=10 loops=50)
"
```

By using the composite indexes idx_universityname_rankingmetric and idx_rankingmetric_universityname, we observe that the actual time was decreasing but the actual cost did not decrease.

Test: idx_order_avgGPA_avgAcademicRep and idx_order_academicRep
(CREATE INDEX idx_order_avgGPA_avgAcademicRep ON AdmissionData(GPA);
CREATE INDEX idx_order_academicRep ON RankingMetric(academicRep);)



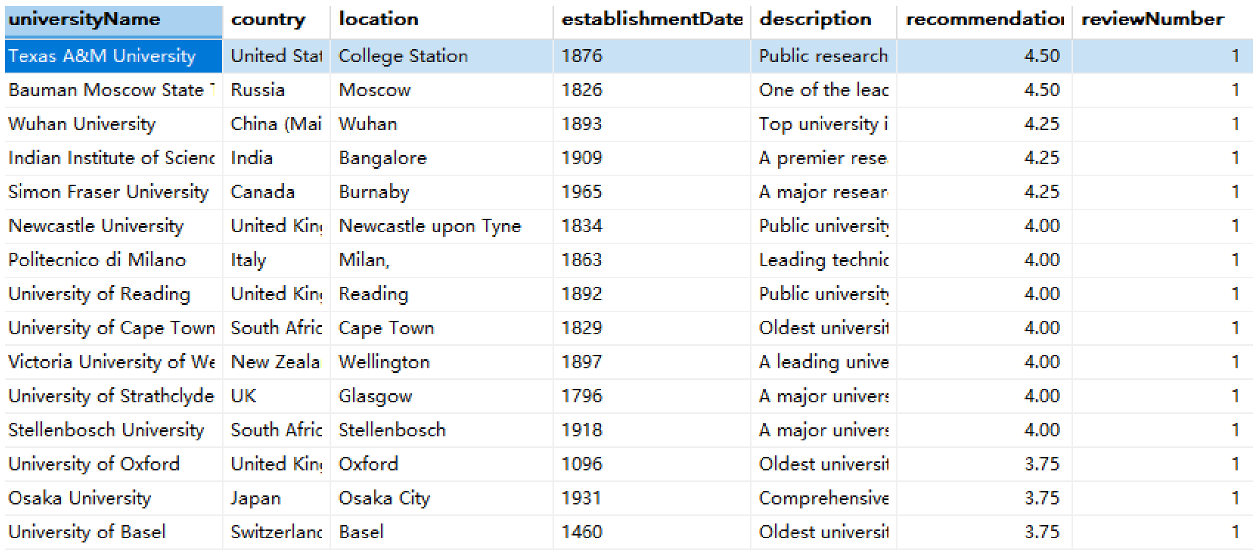
```
EXPLAIN
"-> Limit: 15 row(s)  (actual time=2.573..2.575 rows=15 loops=1)
    -> Sort: avgGPA DESC, avgAcademicRep DESC, limit input to 15 row(s) per chunk  (actual time=2.572..2.573 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=2.447..2.478 rows=211 loops=1)
            -> Aggregate using temporary table  (actual time=2.444..2.444 rows=211 loops=1)
                -> Nested loop inner join  (cost=290.82 rows=729) (actual time=0.359..1.763 rows=514 loops=1)
                    -> Nested loop inner join  (cost=35.65 rows=76) (actual time=0.267..0.497 rows=50 loops=1)
                        -> Covering index lookup on U using university_country (country='United States')  (cost=9.05 rows=38) (actual time=0.234..0.250 rows=38 loops=1)
                        -> Index lookup on R using PRIMARY (universityName=U.universityName)  (cost=0.51 rows=2) (actual time=0.006..0.006 rows=1 loops=38)
                    -> Index lookup on A using AdmissionData_ibfk_2 (universityName=U.universityName)  (cost=2.41 rows=10) (actual time=0.020..0.024 rows=10 loops=50)
"
```

With using these two composite indexes, we can find it is just like what we tested before. These two composite indexes do not help us improve the efficiency of the query. The actual time is even longer than the original one.

So based on what we have tested on this query, we might choose to use the index idx_admission_program to help with our query. Cause it can actually help improve the efficiency of our query

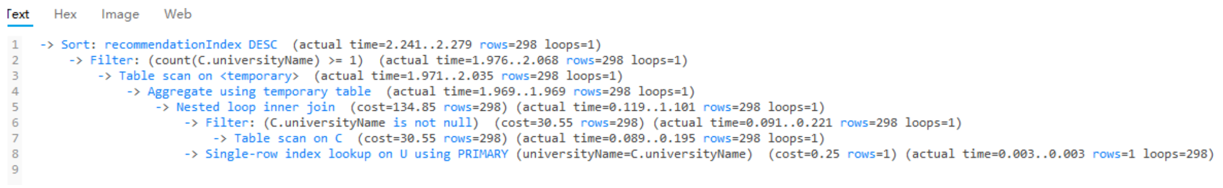
- Get popular universities based on the comments.

```
SELECT
   U.universityName,
   U.country,
   U.location,
   U.establishmentDate,
   U.description,
   ROUND(
   AVG((C.livingEnvironment
    + C.learningAtmosphere
    + C.library
    + C.restaurant) / 4),2)
    AS recommendationIndex,
    COUNT(C.universityName) AS reviewNumber
FROM
    Comment C
JOIN
    University U ON C.universityName = U.universityName
GROUP BY
    U.universityName, U.description, U.country
HAVING
    COUNT(C.universityName) >=1
ORDER BY
    recommendationIndex DESC
 LIMIT 15;
```



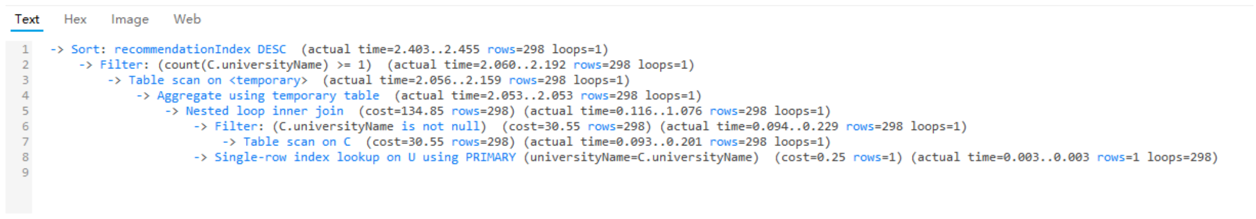
universityName	country	location	establishmentDate	description	recommendatioı	reviewNumber
Texas A&M University	United Stat	College Station	1876	Public research	4.50	1
Bauman Moscow State '	Russia	Moscow	1826	One of the leac	4.50	1
Wuhan University	China (Mai	Wuhan	1893	Top university i	4.25	1
Indian Institute of Scienc	India	Bangalore	1909	A premier rese	4.25	1
Simon Fraser University	Canada	Burnaby	1965	A major resear	4.25	1
Newcastle University	United Kin	Newcastle upon Tyne	1834	Public universit	4.00	1
Politecnico di Milano	Italy	Milan,	1863	Leading technic	4.00	1
University of Reading	United Kin	Reading	1892	Public universit	4.00	1
University of Cape Town	South Afric	Cape Town	1829	Oldest universit	4.00	1
Victoria University of We	New Zeala	Wellington	1897	A leading unive	4.00	1
University of Strathclyde	UK	Glasgow	1796	A major univers	4.00	1
Stellenbosch University	South Afric	Stellenbosch	1918	A major univers	4.00	1
University of Oxford	United Kin	Oxford	1096	Oldest universit	3.75	1
Osaka University	Japan	Osaka City	1931	Comprehensive	3.75	1
University of Basel	Switzerlanc	Basel	1460	Oldest universit	3.75	1

Before adding indexes, the advanced query performance is shown as below:



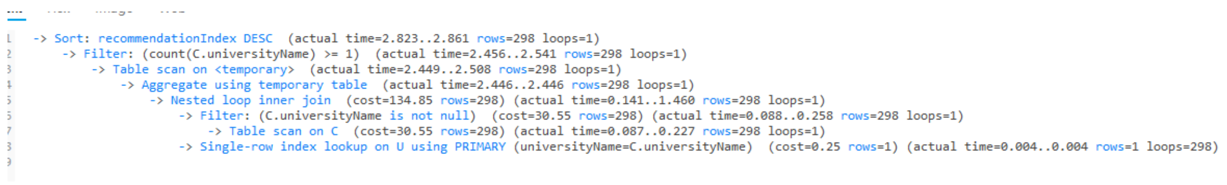
```
-> Sort: recommendationIndex DESC  (actual time=2.241..2.279 rows=298 loops=1)
    -> Filter: (count(C.universityName) >= 1)  (actual time=1.976..2.068 rows=298 loops=1)
        -> Table scan on <temporary>  (actual time=1.971..2.035 rows=298 loops=1)
            -> Aggregate using temporary table  (actual time=1.969..1.969 rows=298 loops=1)
                -> Nested loop inner join  (cost=134.85 rows=298) (actual time=0.119..1.101 rows=298 loops=1)
                    -> Filter: (C.universityName is not null)  (cost=30.55 rows=298) (actual time=0.091..0.221 rows=298 loops=1)
                        -> Table scan on C  (cost=30.55 rows=298) (actual time=0.089..0.195 rows=298 loops=1)
                    -> Single-row index lookup on U using PRIMARY (universityName=C.universityName)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=298)
```

Index on Comment.universityName:



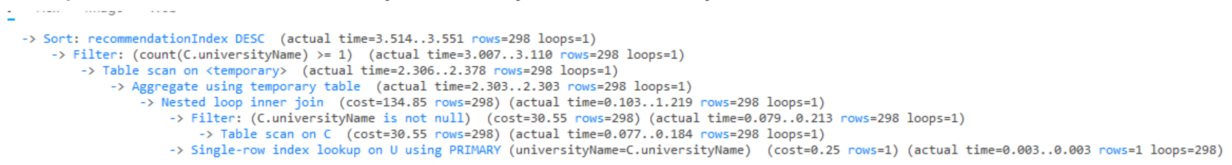
```
-> Sort: recommendationIndex DESC  (actual time=2.403..2.455 rows=298 loops=1)
    -> Filter: (count(C.universityName) >= 1)  (actual time=2.060..2.192 rows=298 loops=1)
        -> Table scan on <temporary>  (actual time=2.056..2.159 rows=298 loops=1)
            -> Aggregate using temporary table  (actual time=2.053..2.053 rows=298 loops=1)
                -> Nested loop inner join  (cost=134.85 rows=298) (actual time=0.116..1.076 rows=298 loops=1)
                    -> Filter: (C.universityName is not null)  (cost=30.55 rows=298) (actual time=0.094..0.229 rows=298 loops=1)
                        -> Table scan on C  (cost=30.55 rows=298) (actual time=0.093..0.201 rows=298 loops=1)
                    -> Single-row index lookup on U using PRIMARY (universityName=C.universityName)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=298)
```

Composite index on University.universityName and country:



```
-> Sort: recommendationIndex DESC  (actual time=2.823..2.861 rows=298 loops=1)
    -> Filter: (count(C.universityName) >= 1)  (actual time=2.456..2.541 rows=298 loops=1)
        -> Table scan on <temporary>  (actual time=2.449..2.508 rows=298 loops=1)
            -> Aggregate using temporary table  (actual time=2.446..2.446 rows=298 loops=1)
                -> Nested loop inner join  (cost=134.85 rows=298) (actual time=0.141..1.460 rows=298 loops=1)
                    -> Filter: (C.universityName is not null)  (cost=30.55 rows=298) (actual time=0.088..0.258 rows=298 loops=1)
                        -> Table scan on C  (cost=30.55 rows=298) (actual time=0.087..0.227 rows=298 loops=1)
                    -> Single-row index lookup on U using PRIMARY (universityName=C.universityName)  (cost=0.25 rows=1) (actual time=0.004..0.004 rows=1 loops=298)
```

Composite Index on University.universityName, country, location



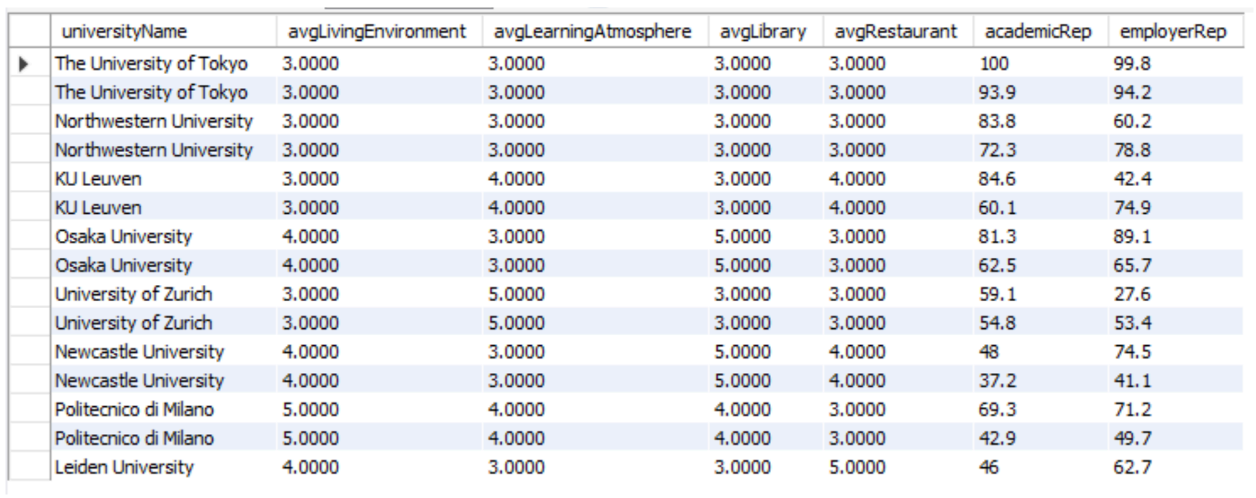
```
-> Sort: recommendationIndex DESC  (actual time=3.514..3.551 rows=298 loops=1)
    -> Filter: (count(C.universityName) >= 1)  (actual time=3.007..3.110 rows=298 loops=1)
        -> Table scan on <temporary>  (actual time=2.306..2.378 rows=298 loops=1)
            -> Aggregate using temporary table  (actual time=2.303..2.303 rows=298 loops=1)
                -> Nested loop inner join  (cost=134.85 rows=298) (actual time=0.103..1.219 rows=298 loops=1)
                    -> Filter: (C.universityName is not null)  (cost=30.55 rows=298) (actual time=0.079..0.213 rows=298 loops=1)
                        -> Table scan on C  (cost=30.55 rows=298) (actual time=0.077..0.184 rows=298 loops=1)
                    -> Single-row index lookup on U using PRIMARY (universityName=C.universityName)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=298)
```

As is shown, the performance has no change, this may be due to the small dataset: The query results show that the dataset is relatively small (only 298 rows involved). And currently there are just several comments for each university. In such cases, the database engine can efficiently perform table scans, joins, and sorts without significant gains from indexes.Besides, Aggregation and Sorting Dominating Query Time: Most of the time is spent in aggregate operations (AVG, COUNT) and sorting, which are not optimized by the indexes added. So we decided not to add index for such functionalities.

- Query to find universities that have above-average user ratings across all key aspects (livingEnvironment, learningAtmosphere, library, restaurant), and also have above-average academic and employer reputation scores, along with the total number of user comments made for each university.

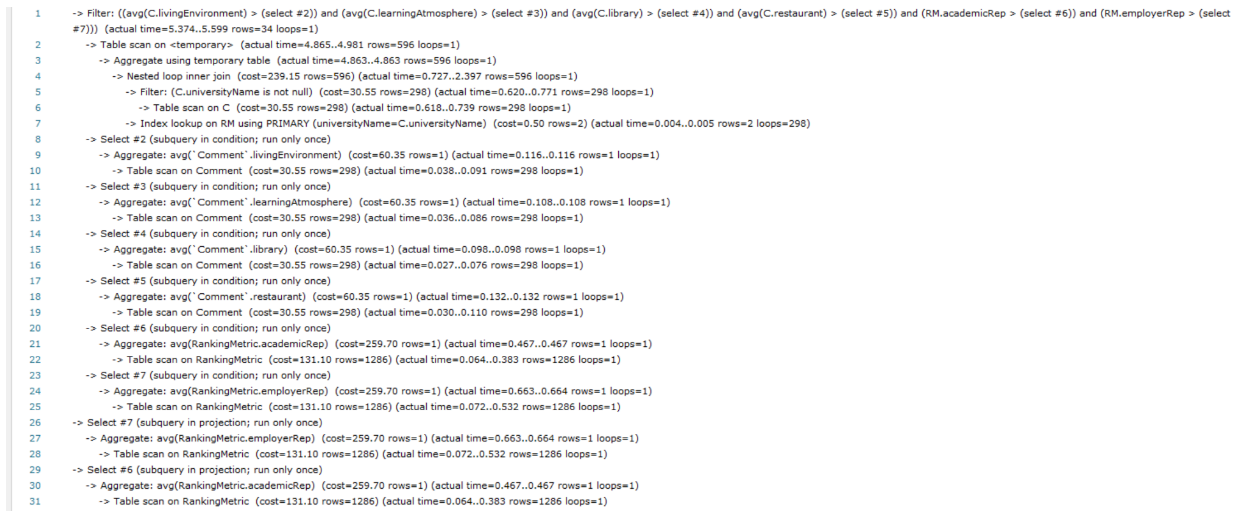
```
SELECT
    C.universityName,
    AVG(C.livingEnvironment) AS avgLivingEnvironment,
    AVG(C.learningAtmosphere) AS avgLearningAtmosphere,
    AVG(C.library) AS avgLibrary,
    AVG(C.restaurant) AS avgRestaurant,
    RM.academicRep,
    RM.employerRep
FROM
    cs411.Comment C
JOIN
    cs411.RankingMetric RM
ON
    C.universityName = RM.universityName
```

```
GROUP BY
    C.universityName, RM.academicRep, RM.employerRep
HAVING
    AVG(C.livingEnvironment) > (
        SELECT AVG(livingEnvironment)
        FROM Comment
    )
    AND AVG(C.learningAtmosphere) > (
        SELECT AVG(learningAtmosphere)
        FROM Comment
    )
    AND AVG(C.library) > (
        SELECT AVG(library)
        FROM Comment
    )
    AND AVG(C.restaurant) > (
        SELECT AVG(restaurant)
        FROM Comment
    )
    AND RM.academicRep > (
        SELECT AVG(academicRep)
        FROM RankingMetric
    )
    AND RM.employerRep > (
        SELECT AVG(employerRep)
        FROM RankingMetric
    )
    LIMIT 15;
```



universityName	avgLivingEnvironment	avgLearningAtmosphere	avgLibrary	avgRestaurant	academicRep	employerRep
The University of Tokyo	3.0000	3.0000	3.0000	3.0000	100	99.8
The University of Tokyo	3.0000	3.0000	3.0000	3.0000	93.9	94.2
Northwestern University	3.0000	3.0000	3.0000	3.0000	83.8	60.2
Northwestern University	3.0000	3.0000	3.0000	3.0000	72.3	78.8
KU Leuven	3.0000	4.0000	3.0000	4.0000	84.6	42.4
KU Leuven	3.0000	4.0000	3.0000	4.0000	60.1	74.9
Osaka University	4.0000	3.0000	5.0000	3.0000	81.3	89.1
Osaka University	4.0000	3.0000	5.0000	3.0000	62.5	65.7
University of Zurich	3.0000	5.0000	3.0000	3.0000	59.1	27.6
University of Zurich	3.0000	5.0000	3.0000	3.0000	54.8	53.4
Newcastle University	4.0000	3.0000	5.0000	4.0000	48	74.5
Newcastle University	4.0000	3.0000	5.0000	4.0000	37.2	41.1
Politecnico di Milano	5.0000	4.0000	4.0000	3.0000	69.3	71.2
Politecnico di Milano	5.0000	4.0000	4.0000	3.0000	42.9	49.7
Leiden University	4.0000	3.0000	3.0000	5.0000	46	62.7

Before adding indexes, the advanced query performance is shown as below:

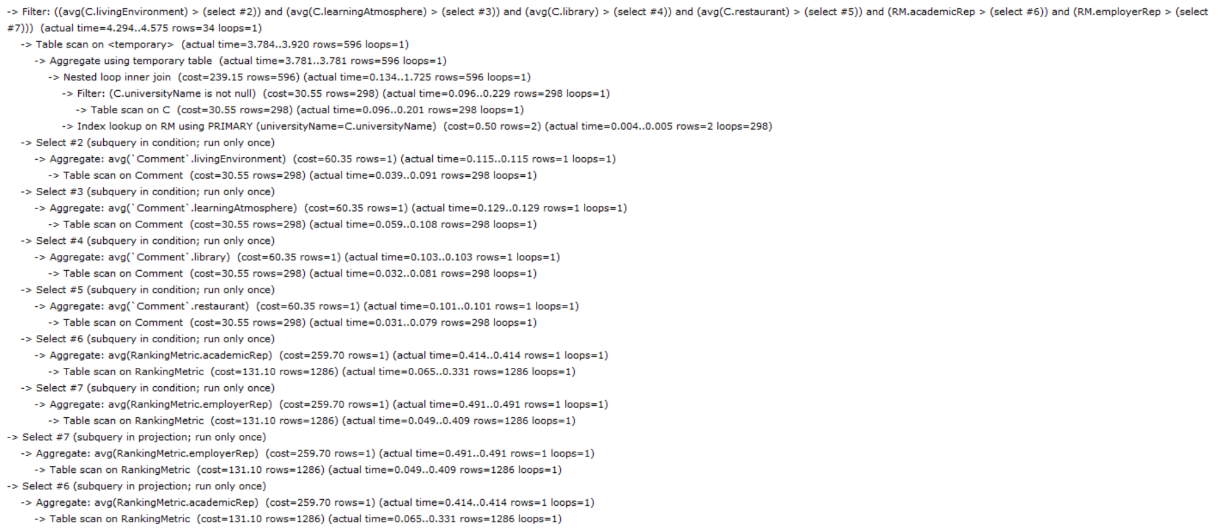


```
-> Filter: ((avg(C.livingEnvironment) > (select #2)) and (avg(C.learningAtmosphere) > (select #3)) and (avg(C.library) > (select #4)) and (avg(C.restaurant) > (select #5)) and (RM.academicRep > (select #6)) and (RM.employerRep > (select
#7)))  (actual time=5.374..5.599 rows=34 loops=1)
    -> Table scan on <temporary>  (actual time=4.865..4.981 rows=596 loops=1)
        -> Aggregate using temporary table  (actual time=4.863..4.863 rows=596 loops=1)
            -> Nested loop inner join  (cost=239.15 rows=596) (actual time=0.727..2.397 rows=596 loops=1)
                -> Filter: (C.universityName is not null)  (cost=30.55 rows=298) (actual time=0.620..0.771 rows=298 loops=1)
                    -> Table scan on C  (cost=30.55 rows=298) (actual time=0.618..0.739 rows=298 loops=1)
                -> Index lookup on RM using PRIMARY (universityName=C.universityName)  (cost=0.50 rows=2) (actual time=0.004..0.005 rows=2 loops=298)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(`Comment`.livingEnvironment)  (cost=60.35 rows=1) (actual time=0.116..0.116 rows=1 loops=1)
            -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.038..0.091 rows=298 loops=1)
    -> Select #3 (subquery in condition; run only once)
        -> Aggregate: avg(`Comment`.learningAtmosphere)  (cost=60.35 rows=1) (actual time=0.108..0.108 rows=1 loops=1)
            -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.036..0.086 rows=298 loops=1)
    -> Select #4 (subquery in condition; run only once)
        -> Aggregate: avg(`Comment`.library)  (cost=60.35 rows=1) (actual time=0.098..0.098 rows=1 loops=1)
            -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.027..0.076 rows=298 loops=1)
    -> Select #5 (subquery in condition; run only once)
        -> Aggregate: avg(`Comment`.restaurant)  (cost=60.35 rows=1) (actual time=0.132..0.132 rows=1 loops=1)
            -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.030..0.110 rows=298 loops=1)
    -> Select #6 (subquery in condition; run only once)
        -> Aggregate: avg(RankingMetric.academicRep)  (cost=259.70 rows=1) (actual time=0.467..0.467 rows=1 loops=1)
            -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.064..0.383 rows=1286 loops=1)
    -> Select #7 (subquery in condition; run only once)
        -> Aggregate: avg(RankingMetric.employerRep)  (cost=259.70 rows=1) (actual time=0.663..0.664 rows=1 loops=1)
            -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.072..0.532 rows=1286 loops=1)
-> Select #7 (subquery in projection; run only once)
    -> Aggregate: avg(RankingMetric.employerRep)  (cost=259.70 rows=1) (actual time=0.663..0.664 rows=1 loops=1)
        -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.072..0.532 rows=1286 loops=1)
-> Select #6 (subquery in projection; run only once)
    -> Aggregate: avg(RankingMetric.academicRep)  (cost=259.70 rows=1) (actual time=0.467..0.467 rows=1 loops=1)
        -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.064..0.383 rows=1286 loops=1)
```

After index JOIN attributes:

CREATE INDEX idx_universityName ON cs411.Comment (universityName);

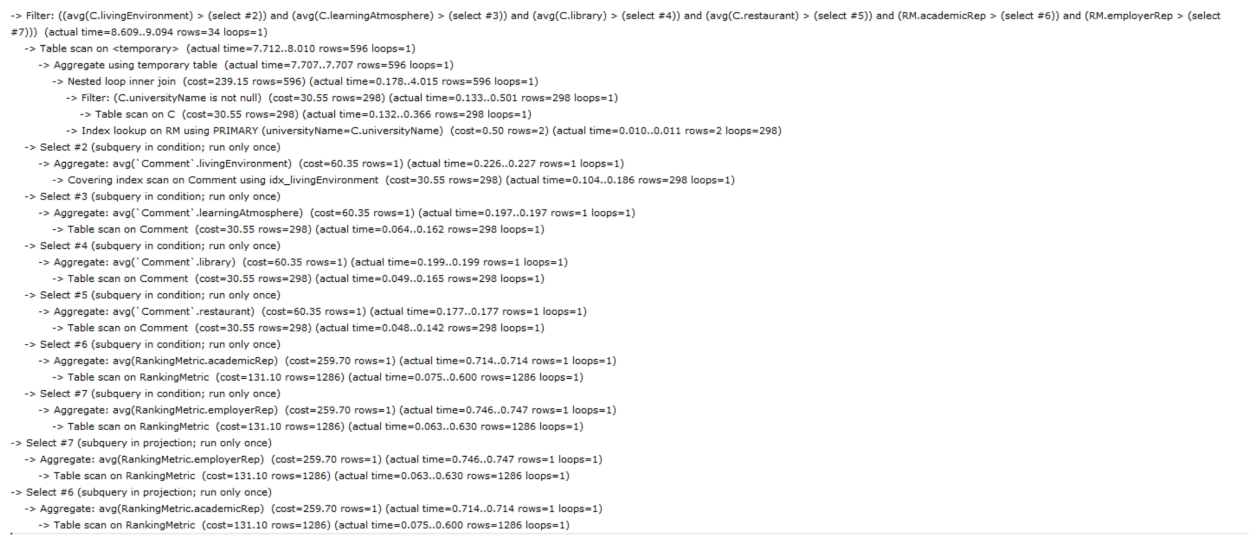
The advanced query performance is shown as below



```
-> Filter: ((avg(C.livingEnvironment) > (select #2)) and (avg(C.learningAtmosphere) > (select #3)) and (avg(C.library) > (select #4)) and (avg(C.restaurant) > (select #5)) and (RM.academicRep > (select #6)) and (RM.employerRep > (select
#7)))  (actual time=4.294..4.575 rows=34 loops=1)
    -> Table scan on <temporary>  (actual time=3.784..3.920 rows=596 loops=1)
        -> Aggregate using temporary table  (actual time=3.781..3.781 rows=596 loops=1)
            -> Nested loop inner join  (cost=239.15 rows=596) (actual time=0.134..1.725 rows=596 loops=1)
                -> Filter: (C.universityName is not null)  (cost=30.55 rows=298) (actual time=0.096..0.229 rows=298 loops=1)
                    -> Table scan on C  (cost=30.55 rows=298) (actual time=0.096..0.201 rows=298 loops=1)
                -> Index lookup on RM using PRIMARY (universityName=C.universityName)  (cost=0.50 rows=2) (actual time=0.004..0.005 rows=2 loops=298)
    -> Select #2 (subquery in condition; run only once)
        -> Aggregate: avg(`Comment`.livingEnvironment)  (cost=60.35 rows=1) (actual time=0.115..0.115 rows=1 loops=1)
            -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.039..0.091 rows=298 loops=1)
    -> Select #3 (subquery in condition; run only once)
        -> Aggregate: avg(`Comment`.learningAtmosphere)  (cost=60.35 rows=1) (actual time=0.129..0.129 rows=1 loops=1)
            -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.059..0.108 rows=298 loops=1)
    -> Select #4 (subquery in condition; run only once)
        -> Aggregate: avg(`Comment`.library)  (cost=60.35 rows=1) (actual time=0.103..0.103 rows=1 loops=1)
            -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.032..0.081 rows=298 loops=1)
    -> Select #5 (subquery in condition; run only once)
        -> Aggregate: avg(`Comment`.restaurant)  (cost=60.35 rows=1) (actual time=0.101..0.101 rows=1 loops=1)
            -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.031..0.079 rows=298 loops=1)
    -> Select #6 (subquery in condition; run only once)
        -> Aggregate: avg(RankingMetric.academicRep)  (cost=259.70 rows=1) (actual time=0.414..0.414 rows=1 loops=1)
            -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.065..0.331 rows=1286 loops=1)
    -> Select #7 (subquery in condition; run only once)
        -> Aggregate: avg(RankingMetric.employerRep)  (cost=259.70 rows=1) (actual time=0.491..0.491 rows=1 loops=1)
            -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.049..0.409 rows=1286 loops=1)
-> Select #7 (subquery in projection; run only once)
    -> Aggregate: avg(RankingMetric.employerRep)  (cost=259.70 rows=1) (actual time=0.491..0.491 rows=1 loops=1)
        -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.049..0.409 rows=1286 loops=1)
-> Select #6 (subquery in projection; run only once)
    -> Aggregate: avg(RankingMetric.academicRep)  (cost=259.70 rows=1) (actual time=0.414..0.414 rows=1 loops=1)
        -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.065..0.331 rows=1286 loops=1)
```

After index HAVING clause:

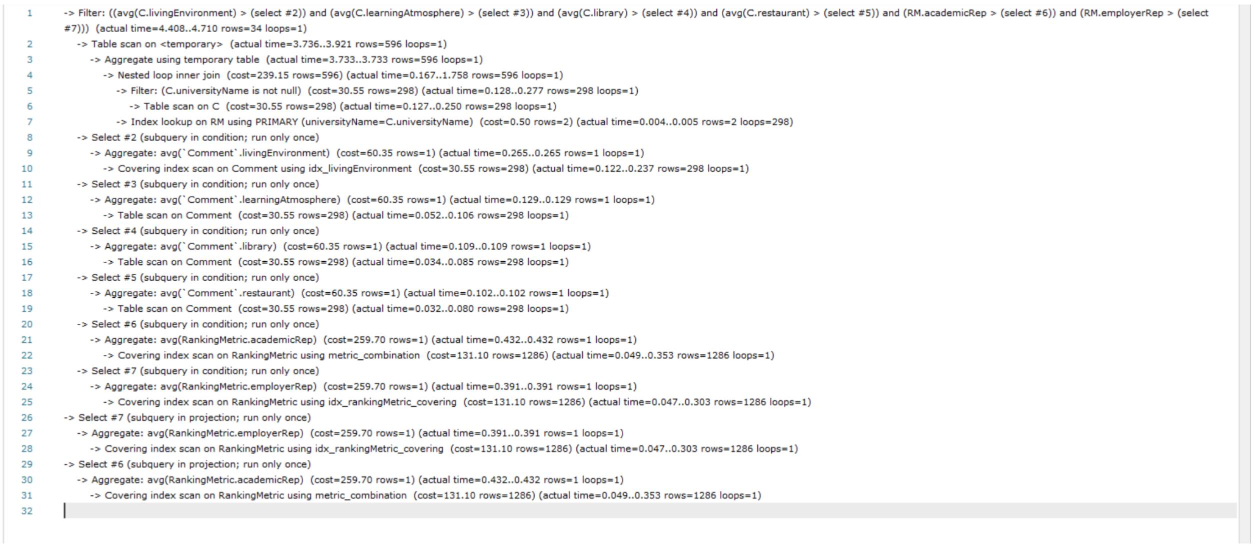
CREATE INDEX idx_livingEnvironment ON cs411.Comment (livingEnvironment);



```
-> Filter: ((avg(C.livingEnvironment) > (select #2)) and (avg(C.learningAtmosphere) > (select #3)) and (avg(C.library) > (select #4)) and (avg(C.restaurant) > (select #5)) and (RM.academicRep > (select #6)) and (RM.employerRep > (select
#7)))  (actual time=8.609..9.094 rows=34 loops=1)
  -> Table scan on <temporary>  (actual time=7.712..8.010 rows=596 loops=1)
    -> Aggregate using temporary table  (actual time=7.707..7.707 rows=596 loops=1)
      -> Nested loop inner join  (cost=239.15 rows=596) (actual time=0.178..4.015 rows=596 loops=1)
        -> Filter: (C.universityName is not null)  (cost=30.55 rows=298) (actual time=0.133..0.501 rows=298 loops=1)
          -> Table scan on C  (cost=30.55 rows=298) (actual time=0.132..0.366 rows=298 loops=1)
        -> Index lookup on RM using PRIMARY (universityName=C.universityName)  (cost=0.50 rows=2) (actual time=0.010..0.011 rows=2 loops=298)
  -> Select #2 (subquery in condition; run only once)
    -> Aggregate: avg(`Comment`.livingEnvironment)  (cost=60.35 rows=1) (actual time=0.226..0.227 rows=1 loops=1)
      -> Covering index scan on Comment using idx_livingEnvironment  (cost=30.55 rows=298) (actual time=0.104..0.186 rows=298 loops=1)
  -> Select #3 (subquery in condition; run only once)
    -> Aggregate: avg(`Comment`.learningAtmosphere)  (cost=60.35 rows=1) (actual time=0.197..0.197 rows=1 loops=1)
      -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.064..0.162 rows=298 loops=1)
  -> Select #4 (subquery in condition; run only once)
    -> Aggregate: avg(`Comment`.library)  (cost=60.35 rows=1) (actual time=0.199..0.199 rows=1 loops=1)
      -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.049..0.165 rows=298 loops=1)
  -> Select #5 (subquery in condition; run only once)
    -> Aggregate: avg(`Comment`.restaurant)  (cost=60.35 rows=1) (actual time=0.177..0.177 rows=1 loops=1)
      -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.048..0.142 rows=298 loops=1)
  -> Select #6 (subquery in condition; run only once)
    -> Aggregate: avg(RankingMetric.academicRep)  (cost=259.70 rows=1) (actual time=0.714..0.714 rows=1 loops=1)
      -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.075..0.600 rows=1286 loops=1)
  -> Select #7 (subquery in condition; run only once)
    -> Aggregate: avg(RankingMetric.employerRep)  (cost=259.70 rows=1) (actual time=0.746..0.747 rows=1 loops=1)
      -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.063..0.630 rows=1286 loops=1)
-> Select #7 (subquery in projection; run only once)
  -> Aggregate: avg(RankingMetric.employerRep)  (cost=259.70 rows=1) (actual time=0.746..0.747 rows=1 loops=1)
    -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.063..0.630 rows=1286 loops=1)
-> Select #6 (subquery in projection; run only once)
  -> Aggregate: avg(RankingMetric.academicRep)  (cost=259.70 rows=1) (actual time=0.714..0.714 rows=1 loops=1)
    -> Table scan on RankingMetric  (cost=131.10 rows=1286) (actual time=0.075..0.600 rows=1286 loops=1)
```

And I also add index on universityName,academicRep and employerRep together:

CREATE INDEX idx_rankingMetric_covering ON cs411.RankingMetric (universityName, academicRep, employerRep);



```
-> Filter: ((avg(C.livingEnvironment) > (select #2)) and (avg(C.learningAtmosphere) > (select #3)) and (avg(C.library) > (select #4)) and (avg(C.restaurant) > (select #5)) and (RM.academicRep > (select #6)) and (RM.employerRep > (select
#7)))  (actual time=4.408..4.710 rows=34 loops=1)
  -> Table scan on <temporary>  (actual time=3.736..3.921 rows=596 loops=1)
    -> Aggregate using temporary table  (actual time=3.733..3.733 rows=596 loops=1)
      -> Nested loop inner join  (cost=239.15 rows=596) (actual time=0.167..1.758 rows=596 loops=1)
        -> Filter: (C.universityName is not null)  (cost=30.55 rows=298) (actual time=0.128..0.277 rows=298 loops=1)
          -> Table scan on C  (cost=30.55 rows=298) (actual time=0.127..0.250 rows=298 loops=1)
        -> Index lookup on RM using PRIMARY (universityName=C.universityName)  (cost=0.50 rows=2) (actual time=0.004..0.005 rows=2 loops=298)
  -> Select #2 (subquery in condition; run only once)
    -> Aggregate: avg(`Comment`.livingEnvironment)  (cost=60.35 rows=1) (actual time=0.265..0.265 rows=1 loops=1)
      -> Covering index scan on Comment using idx_livingEnvironment  (cost=30.55 rows=298) (actual time=0.122..0.237 rows=298 loops=1)
  -> Select #3 (subquery in condition; run only once)
    -> Aggregate: avg(`Comment`.learningAtmosphere)  (cost=60.35 rows=1) (actual time=0.129..0.129 rows=1 loops=1)
      -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.052..0.106 rows=298 loops=1)
  -> Select #4 (subquery in condition; run only once)
    -> Aggregate: avg(`Comment`.library)  (cost=60.35 rows=1) (actual time=0.109..0.109 rows=1 loops=1)
      -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.034..0.085 rows=298 loops=1)
  -> Select #5 (subquery in condition; run only once)
    -> Aggregate: avg(`Comment`.restaurant)  (cost=60.35 rows=1) (actual time=0.102..0.102 rows=1 loops=1)
      -> Table scan on Comment  (cost=30.55 rows=298) (actual time=0.032..0.080 rows=298 loops=1)
  -> Select #6 (subquery in condition; run only once)
    -> Aggregate: avg(RankingMetric.academicRep)  (cost=259.70 rows=1) (actual time=0.432..0.432 rows=1 loops=1)
      -> Covering index scan on RankingMetric using metric_combination  (cost=131.10 rows=1286) (actual time=0.049..0.353 rows=1286 loops=1)
  -> Select #7 (subquery in condition; run only once)
    -> Aggregate: avg(RankingMetric.employerRep)  (cost=259.70 rows=1) (actual time=0.391..0.391 rows=1 loops=1)
      -> Covering index scan on RankingMetric using idx_rankingMetric_covering  (cost=131.10 rows=1286) (actual time=0.047..0.303 rows=1286 loops=1)
-> Select #7 (subquery in projection; run only once)
  -> Aggregate: avg(RankingMetric.employerRep)  (cost=259.70 rows=1) (actual time=0.391..0.391 rows=1 loops=1)
    -> Covering index scan on RankingMetric using idx_rankingMetric_covering  (cost=131.10 rows=1286) (actual time=0.047..0.303 rows=1286 loops=1)
-> Select #6 (subquery in projection; run only once)
  -> Aggregate: avg(RankingMetric.academicRep)  (cost=259.70 rows=1) (actual time=0.432..0.432 rows=1 loops=1)
    -> Covering index scan on RankingMetric using metric_combination  (cost=131.10 rows=1286) (actual time=0.049..0.353 rows=1286 loops=1)
```

As is shown, the performance has no change. There may be two reasons: first is Indexing is most effective when the indexed columns have high selectivity, meaning they contain many unique values. If the universityName or columns in the HAVING clause contain only a few distinct values (e.g., many rows have the same value), the database may determine that it's more efficient to perform a full table scan rather than using the index; second is universityName appears in many rows (low uniqueness), then using an index for the join may not provide much benefit compared to a sequential scan, especially for larger tables.