a. Create a report with the following topics. The report would be graded by completeness and correctness.

   i. Briefly describe what the project accomplished.
   ii. Discuss the usefulness of your project, i.e. what real problem you solved.
   iii. Discuss the data in your database
   iv. Include your updated ER Diagram,
   v. Include your final database schema (DDLs) and index design analysis
   vi. Briefly discuss from where you collected data and how you did it (if crawling is automated, explain how and what tools were used)
   vii. Briefly discuss your application design and the features involved. Clearly list the functionality of your application (feature specs)
   viii. Include your advanced database program. Discuss your choice for the advanced database program and how it is suitable for your application. For example, if you choose transaction+trigger, explain how this choice is suitable for your application, the isolation level of your transaction, etc.
   ix. List and briefly explain the dataflow, i.e. the steps that occur between a user entering the data on the screen and the output that occurs (you can insert a set of screenshots).
   x. Describe one technical challenge that the team encountered. This should be sufficiently detailed such that another future team could use this as helpful advice if they were to start a similar project or where to maintain your project. Say you created a very robust crawler - share your knowledge. You learned how to visualize a graph and make an interactive interface for it - teach all! Know how to minimize time building a mobile app - describe!
   xi. State if everything went according to the initial development plan and proposed specifications, if not - why?!
   xii. Describe the final division of labor and how well you managed teamwork.

Most Valuable Professor

A. Project Accomplishment:-

Our project is a website that makes it easy for UIUC students to choose classes. Students are able to filter/search courses and professors based on parameters like general education tags, gpa threshold, year range, average rating, and average hours of work. Students can create accounts and login to the website. This allows students to rate their professor/courses and track their search history. The GPA data is based entirely on data reported by the university instead of user reported.

B. Project Usefulness:-

This project is useful in the way that it provides a data-based approach to providing information about what to expect from a professor/course at UIUC, along with the ability to search gen ed courses with tags specific to the university. Similar applications like RateMyProfessor do not offer as much detail about courses or professors and do not have statistics.

C. Data in our Database:-

We have 9 tables in our database.

RegisteredUser: It contains all the NetIds of the students and their Passwords in order to log in and leave any rating for a professor/course. NetID is the primary key.. We will not allow deletion of the user from this table.

AdminUser: It contains the NetId and Password of the Admin-rank users (aka us). Admin rank allows us to use the built-in CRUD commands on our website to manipulate the database directly on the website. NetID is the primary key.

CourseProperties: It has the CourseTitle, CourseDescription, Prerequisites, LinkedSections, CourseWebsiteLink, CourseNumber, Subject and CourseTag. Here we assign CourseID to each entry and make it a primary key because CourseNumber and CourseSubject may not be unique for all classes. For instance, there could be 411 CourseNumber for CS as well as for the Math department. And CourseSubject could have many different courses under it. For instance, CS CourseSubject could have 411 and 241 under it. As a result, we chose CourseID as our primary key as it will be unique to all class's CourseNumber and CourseSubject combination. Prerequisites will be used as CSV values of various classes that are needed to take this class.

CourseStats: It contains the CourseID, Year/Term that class was taught, the grade distribution (A+,A,A-,B+,B,B-,C+,C,C-,D+,D,D-,F,W), and the Professor that taught that class. We use CourseID as a foreign key to the CourseProperties table.

Feedback: It maintains the NetID of the user, CourseID of the course, Professor name, Rating/Time Consumption/Comment that the user gave, and TimeStamp of the rating. Here we have NetID as the foreign key because the same user can leave ratings for many classes. Also, we have the CourseID to CourseProperties table as our foreign key because multiple users can leave ratings for the same course.

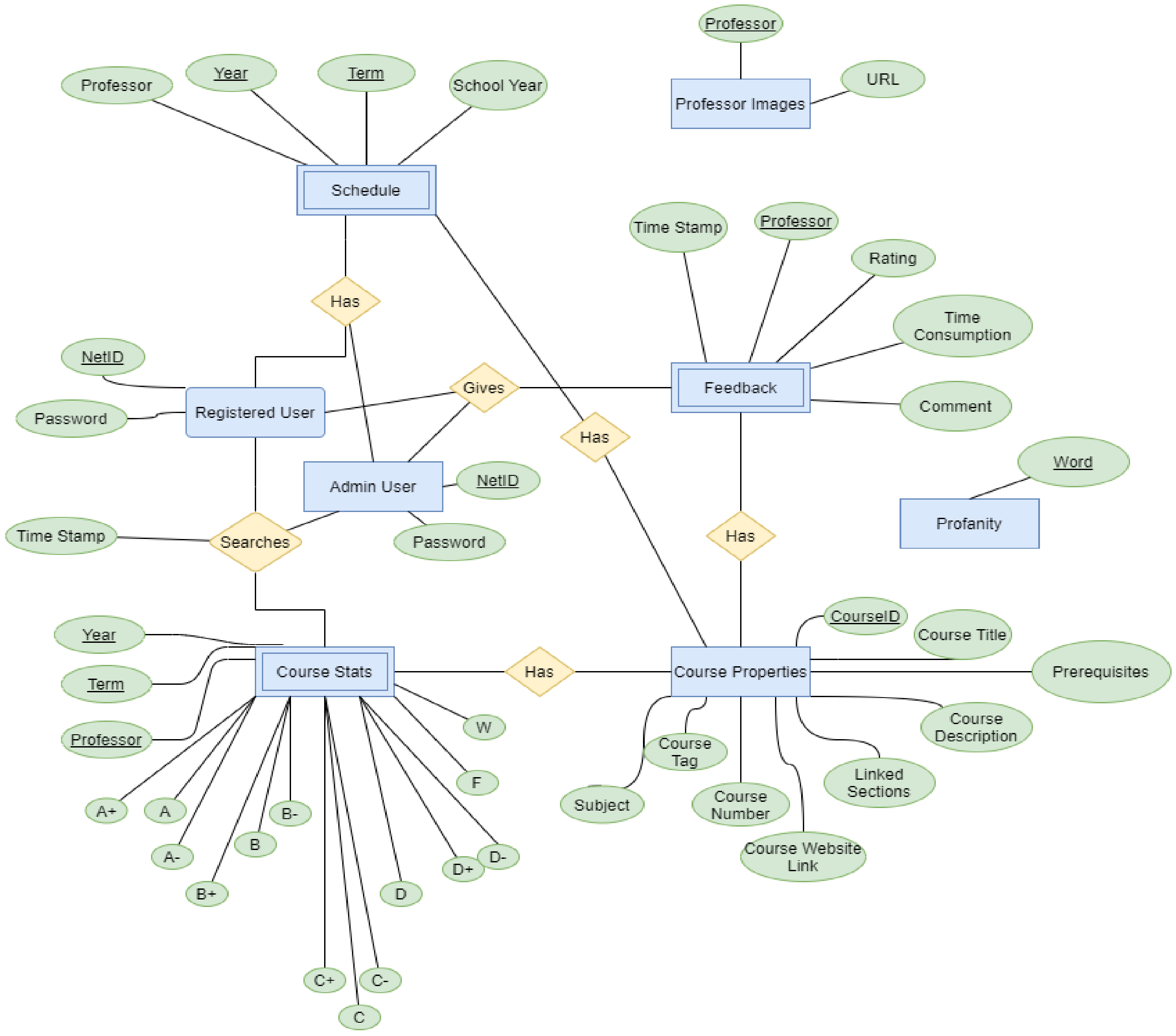
Schedule: It contains NetID, CourseID, Professor, Year, Term, and SchoolYear. CourseID will be used to reference a specific course, Professor will be used to reference a specific professor that the student took, Year/Term will be used to store the semester the course was taken, and SchoolYear will be used to help other students to see the schedule other students of the same school year. Here, CourseID is a foreign key to the CourseProperties table as multiple users can sign up for the same course. NetID is also a foreign key.

SearchHistory: It contains the NetID, CourseID, and Professor. The NetID is used to identify which student made the search, the Professor and CourseID are used to identify the professor/course page the user clicked on. All of the fields are primary keys. NetID and CourseID are foreign keys.

Profanity: It contains profane words that are to be filtered out if matched when a user submits a feedback. This table is used by our trigger on insert into feedback or update on feedback. The primary key and the only column is Word.

ProfessorImages: It contains all the urls of images of professors at UIUC. This is used by our website to display the image of a professor when a course/professor link is opened. The primary key is the Professor and the only other column is the URL.

D. Updated ER Diagram:-



E. DDLS and Index Design Analysis:-

1) CREATE TABLE `AdminUser` (
```
  `NetID` varchar(20) NOT NULL,
  `Password` varchar(100) NOT NULL,
  PRIMARY KEY (`NetID`)
)
```

2) CREATE TABLE `CourseProperties` (
```
  `CourseID` int(11) NOT NULL,
  `CourseTitle` varchar(100) NOT NULL,
  `CourseDescription` varchar(1500) DEFAULT NULL,
  `Prerequisites` varchar(750) DEFAULT NULL,
  `LinkedSections` varchar(750) DEFAULT NULL,
  `CourseWebsiteLink` varchar(200) DEFAULT NULL,
  `CourseNumber` int(11) NOT NULL,
  `CourseTag` varchar(100) DEFAULT NULL,
  `Subject` varchar(10) NOT NULL,
  PRIMARY KEY (`CourseID`),
  KEY `idx_courseTitle` (`CourseTitle`)
)
```

3) CREATE TABLE `CourseStats` (
```
  `CourseID` int(11) NOT NULL,
  `Year` varchar(10) NOT NULL,
  `Term` varchar(10) NOT NULL,
  `Professor` varchar(100) NOT NULL,
  `AP` int(11) DEFAULT NULL,
  `A` int(11) DEFAULT NULL,
  `AM` int(11) DEFAULT NULL,
  `BP` int(11) DEFAULT NULL,
  `B` int(11) DEFAULT NULL,
  `BM` int(11) DEFAULT NULL,
  `CP` int(11) DEFAULT NULL,
  `C` int(11) DEFAULT NULL,
  `CM` int(11) DEFAULT NULL,
  `DP` int(11) DEFAULT NULL,
  `D` int(11) DEFAULT NULL,
  `DM` int(11) DEFAULT NULL,
  `F` int(11) DEFAULT NULL,
  `W` int(11) DEFAULT NULL,
  KEY `CourseID` (`CourseID`),
  CONSTRAINT `CourseStats_ibfk_1` FOREIGN KEY (`CourseID`)
REFERENCES `CourseProperties` (`CourseID`) ON DELETE CASCADE
ON UPDATE CASCADE
)
```

4) CREATE TABLE `Feedback` (

```
  `NetID` varchar(20) NOT NULL,
  `CourseID` int(11) NOT NULL,
  `Professor` varchar(100) NOT NULL,
  `Rating` int(11) NOT NULL,
  `TimeConsumption` varchar(20) DEFAULT NULL,
  `Comment` varchar(500) DEFAULT NULL,
  `TimeStamp` varchar(100) NOT NULL,
  PRIMARY KEY (`NetID`,`CourseID`,`Professor`),
  KEY `CourseID` (`CourseID`),
  CONSTRAINT `Feedback_ibfk_1` FOREIGN KEY (`NetID`)
REFERENCES `RegisteredUser` (`NetID`) ON DELETE CASCADE ON
UPDATE CASCADE,
  CONSTRAINT `Feedback_ibfk_2` FOREIGN KEY (`CourseID`)
REFERENCES `CourseProperties` (`CourseID`) ON DELETE CASCADE
ON UPDATE CASCADE
)
```

5)
```
CREATE TABLE `Profanity` (
  `Word` varchar(50) NOT NULL,
  PRIMARY KEY (`Word`)
)
```

6)
```
CREATE TABLE `ProfessorImages` (
  `Professor` varchar(100) NOT NULL,
  `URL` mediumtext NOT NULL,
  PRIMARY KEY (`Professor`)
)
```

7)
```
CREATE TABLE `RegisteredUser` (
  `NetID` varchar(20) NOT NULL,
  `Password` varchar(100) NOT NULL,
  PRIMARY KEY (`NetID`)
)
```

8) CREATE TABLE `Schedule` (

```
  `NetID` varchar(20) NOT NULL,
  `Professor` varchar(100) NOT NULL,
  `CourseID` int(11) NOT NULL,
  `Year` int(11) NOT NULL,
  `Term` varchar(10) NOT NULL,
  `SchoolYear` varchar(10) NOT NULL,
  PRIMARY KEY (`NetID`,`CourseID`,`Year`,`Term`),
  KEY `CourseID` (`CourseID`),
  CONSTRAINT `Schedule_ibfk_1` FOREIGN KEY (`CourseID`)
REFERENCES `CourseProperties` (`CourseID`) ON DELETE CASCADE
ON UPDATE CASCADE,
  CONSTRAINT `Schedule_ibfk_2` FOREIGN KEY (`NetID`)
REFERENCES `RegisteredUser` (`NetID`) ON DELETE CASCADE ON
UPDATE CASCADE
)
```

9) CREATE TABLE `SearchHistory` (

```
  `NetID` varchar(20) NOT NULL,
  `CourseID` int(11) NOT NULL,
  `Professor` varchar(100) NOT NULL,
  PRIMARY KEY (`NetID`,`CourseID`,`Professor`),
  KEY `CourseID` (`CourseID`),
  CONSTRAINT `SearchHistory_ibfk_1` FOREIGN KEY (`NetID`)
REFERENCES `RegisteredUser` (`NetID`) ON DELETE CASCADE ON
UPDATE CASCADE,
```

CONSTRAINT `SearchHistory_ibfk_2` FOREIGN KEY (`CourseID`) REFERENCES `CourseProperties` (`CourseID`) ON DELETE CASCADE ON UPDATE CASCADE
)

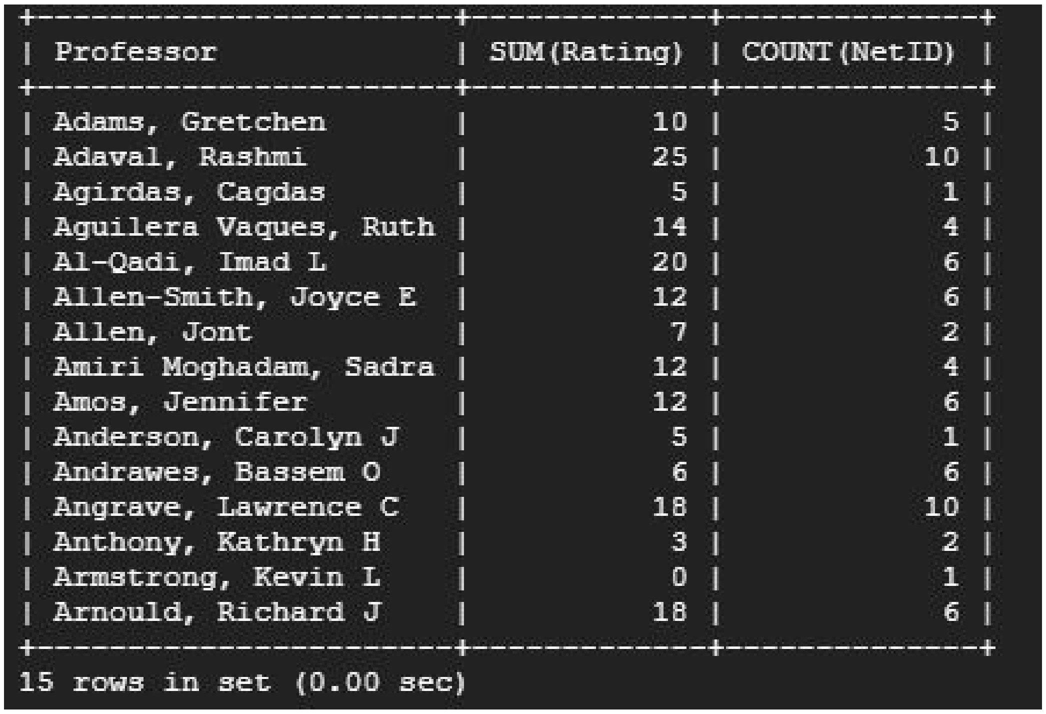
Index Design:

We created indices on the Professor column in the Feedback table, Professor column in the SearchHistory table, Professor column in CourseStats, and CourseTitle in CourseProperties table.

Index Analysis:

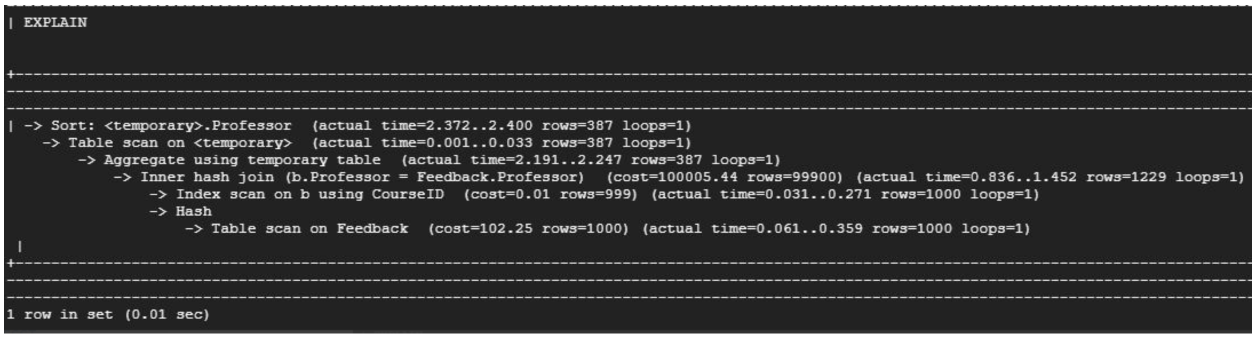
We performed our analysis of our indices on the four advanced queries below:

1) SELECT a.Professor, SUM(Rating), COUNT(NetID) FROM (SELECT Professor, Rating FROM Feedback) a JOIN SearchHistory b ON a.Professor = b.Professor GROUP BY a.Professor ORDER BY a.Professor;



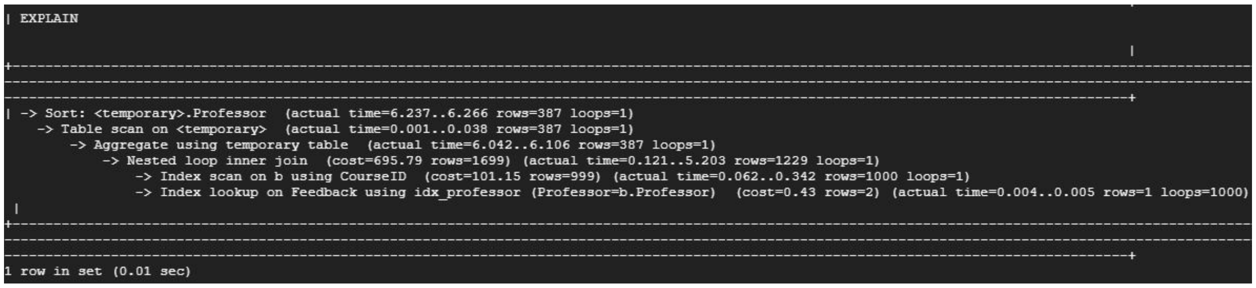
```
+-----------------------+-------------+--------------+
| Professor             | SUM(Rating) | COUNT(NetID) |
+-----------------------+-------------+--------------+
| Adams, Gretchen       |          10 |            5 |
| Adaval, Rashmi        |          25 |           10 |
| Agirdas, Cagdas       |           5 |            1 |
| Aguilera Vaques, Ruth |          14 |            4 |
| Al-Qadi, Imad L       |          20 |            6 |
| Allen-Smith, Joyce E  |          12 |            6 |
| Allen, Jont           |           7 |            2 |
| Amiri Moghadam, Sadra |          12 |            4 |
| Amos, Jennifer        |          12 |            6 |
| Anderson, Carolyn J   |           5 |            1 |
| Andrawes, Bassem O    |           6 |            6 |
| Angrave, Lawrence C   |          18 |           10 |
| Anthony, Kathryn H    |           3 |            2 |
| Armstrong, Kevin L    |           0 |            1 |
| Arnould, Richard J    |          18 |            6 |
+-----------------------+-------------+--------------+
15 rows in set (0.00 sec)
```

Without any indices the query ran in 0.01 seconds.



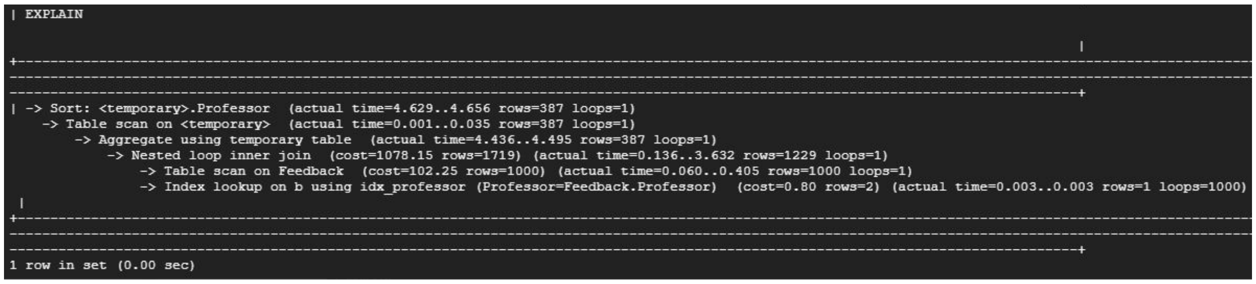
```
| EXPLAIN

+------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
| -> Sort: <temporary>.Professor  (actual time=2.372..2.400 rows=387 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.033 rows=387 loops=1)
        -> Aggregate using temporary table  (actual time=2.191..2.247 rows=387 loops=1)
            -> Inner hash join (b.Professor = Feedback.Professor)  (cost=100005.44 rows=99900) (actual time=0.836..1.452 rows=1229 loops=1)
                -> Index scan on b using CourseID  (cost=0.01 rows=999) (actual time=0.031..0.271 rows=1000 loops=1)
                -> Hash
                    -> Table scan on Feedback  (cost=102.25 rows=1000) (actual time=0.061..0.359 rows=1000 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

After an index was created on the professor column in the Feedback Table:



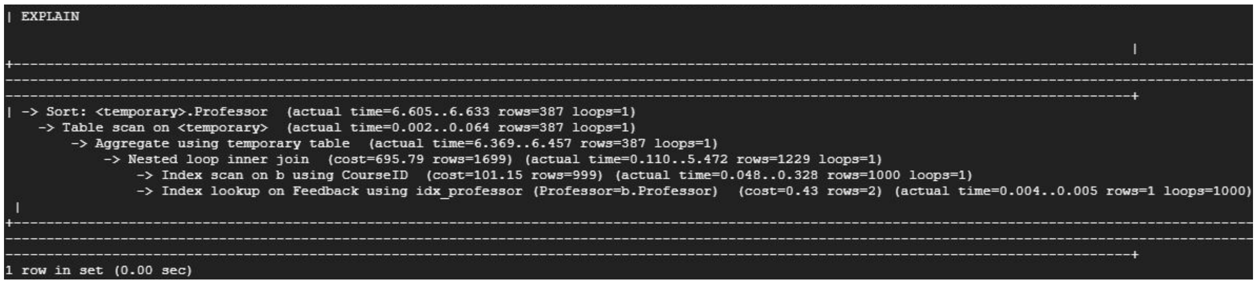
```
| EXPLAIN
                                                                                                                    |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------+
| -> Sort: <temporary>.Professor  (actual time=6.237..6.266 rows=387 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.038 rows=387 loops=1)
        -> Aggregate using temporary table  (actual time=6.042..6.106 rows=387 loops=1)
            -> Nested loop inner join  (cost=695.79 rows=1699) (actual time=0.121..5.203 rows=1229 loops=1)
                -> Index scan on b using CourseID  (cost=101.15 rows=999) (actual time=0.062..0.342 rows=1000 loops=1)
                -> Index lookup on Feedback using idx_professor (Professor=b.Professor)  (cost=0.43 rows=2) (actual time=0.004..0.005 rows=1 loops=1000)
 |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

After an index was created on the professor column in the SearchHistory Table:

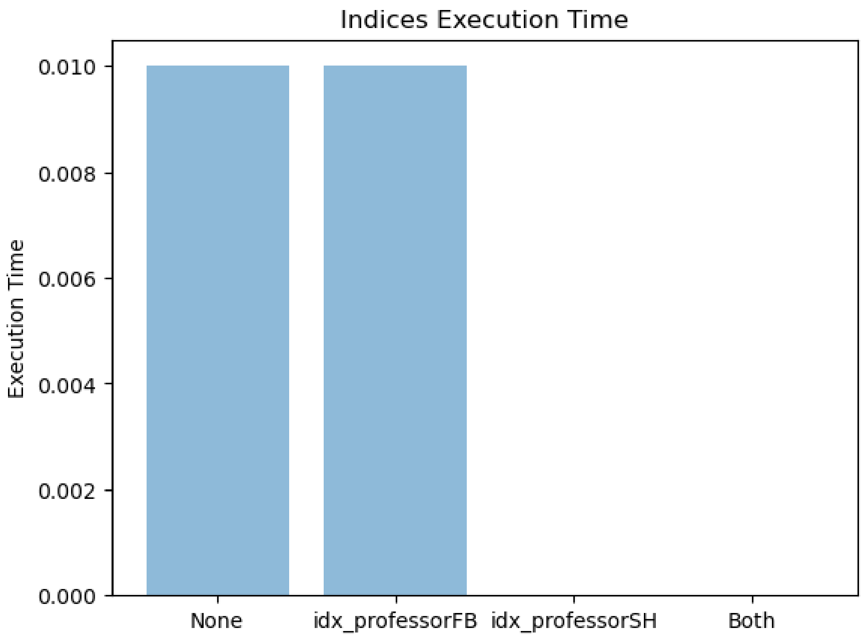


```
| EXPLAIN
                                                                                                                |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------+
| -> Sort: <temporary>.Professor  (actual time=4.629..4.656 rows=387 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.035 rows=387 loops=1)
        -> Aggregate using temporary table  (actual time=4.436..4.495 rows=387 loops=1)
            -> Nested loop inner join  (cost=1078.15 rows=1719) (actual time=0.136..3.632 rows=1229 loops=1)
                -> Table scan on Feedback  (cost=102.25 rows=1000) (actual time=0.060..0.405 rows=1000 loops=1)
                -> Index lookup on b using idx_professor (Professor=Feedback.Professor)  (cost=0.80 rows=2) (actual time=0.003..0.003 rows=1 loops=1000)
 |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

After both indices:

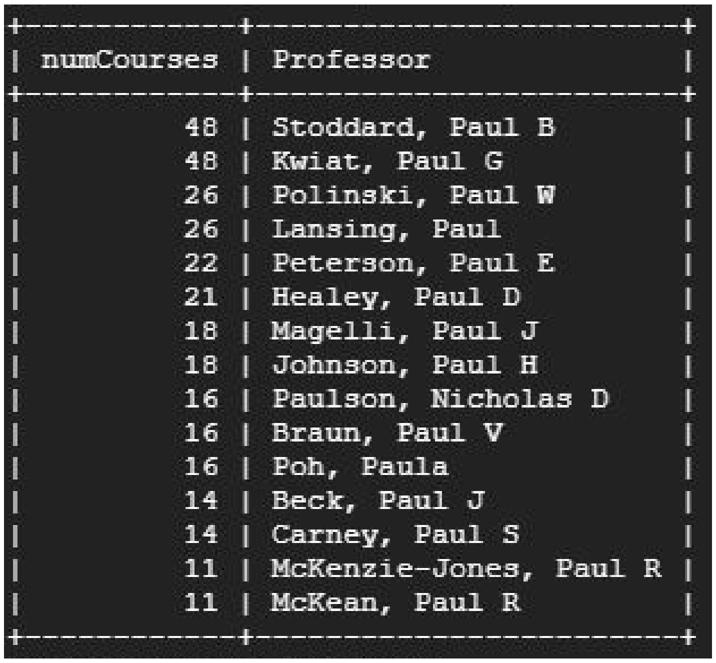


```
| EXPLAIN
                                                                                                                    |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------+
| -> Sort: <temporary>.Professor  (actual time=6.605..6.633 rows=387 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.064 rows=387 loops=1)
        -> Aggregate using temporary table  (actual time=6.369..6.457 rows=387 loops=1)
            -> Nested loop inner join  (cost=695.79 rows=1699) (actual time=0.110..5.472 rows=1229 loops=1)
                -> Index scan on b using CourseID  (cost=101.15 rows=999) (actual time=0.048..0.328 rows=1000 loops=1)
                -> Index lookup on Feedback using idx_professor (Professor=b.Professor)  (cost=0.43 rows=2) (actual time=0.004..0.005 rows=1 loops=1000)
 |
+------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```



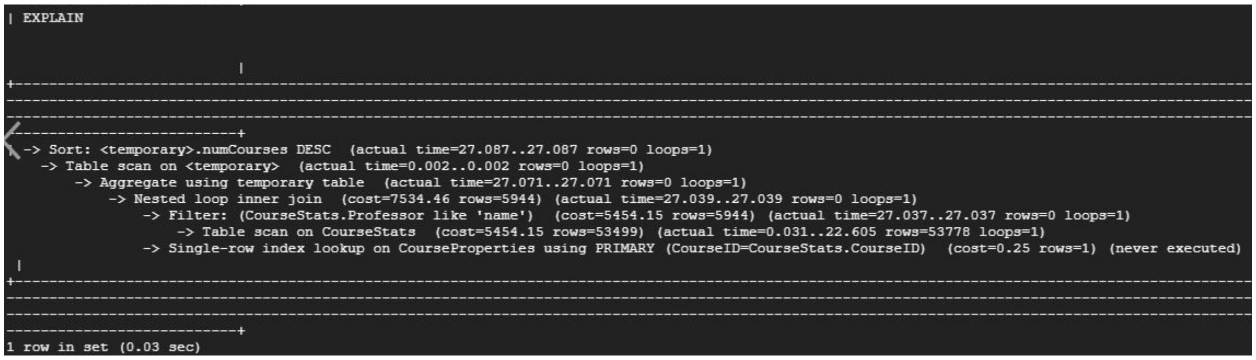
Conclusion: Creating an index on the professor column in the Feedback table did not have any benefit on the execution time, but having an index on the professor column in the SearchHistory table gave a boost to the execution time. It might be because of a join between Feedback and SearchHistory table. Finally, adding both the index on SearchHistory and Feedback table will be helpful in the future as our database grows. Currently there are only 1000 rows so it might not show a significant improvement, but later it will show improvements as we are using group by on professor column on Feedback table and join on professor with SearchHistory table.

2) SELECT COUNT(CourseID) as numCourses, Professor FROM (SELECT CourseID, Professor FROM CourseProperties NATURAL JOIN CourseStats WHERE Professor LIKE "name") as joined_table GROUP BY Professor ORDER BY numCourses DESC;



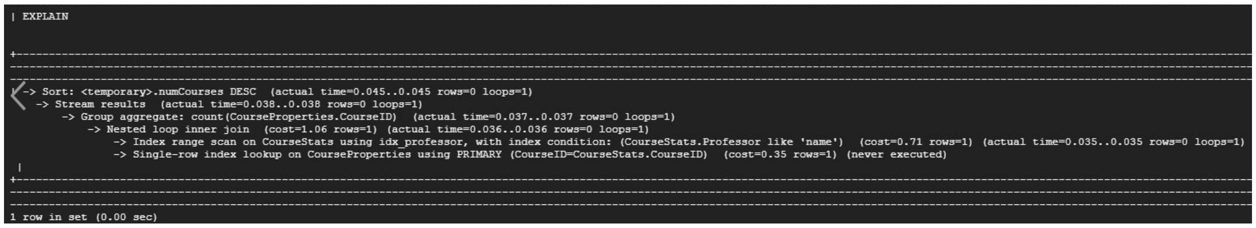
```
+-------------+-------------------------+
| numCourses | Professor               |
+-------------+-------------------------+
|          48 | Stoddard, Paul B        |
|          48 | Kwiat, Paul G           |
|          26 | Polinski, Paul W        |
|          26 | Lansing, Paul           |
|          22 | Peterson, Paul E        |
|          21 | Healey, Paul D          |
|          18 | Magelli, Paul J         |
|          18 | Johnson, Paul H         |
|          16 | Paulson, Nicholas D     |
|          16 | Braun, Paul V           |
|          16 | Poh, Paula              |
|          14 | Beck, Paul J            |
|          14 | Carney, Paul S          |
|          11 | McKenzie-Jones, Paul R  |
|          11 | McKean, Paul R          |
+-------------+-------------------------+
```

Without any indices this query ran in 0.03 seconds.

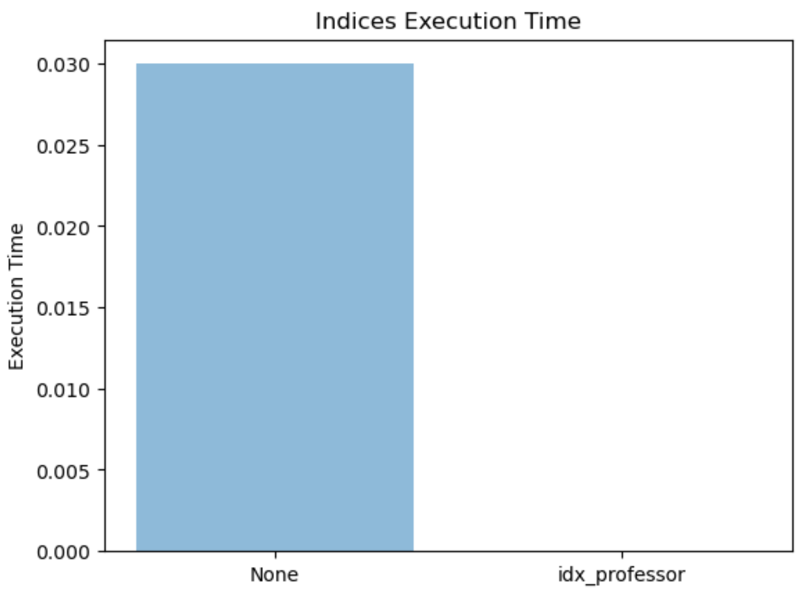


```
| EXPLAIN


                         |
+--------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------+
| -> Sort: <temporary>.numCourses DESC  (actual time=27.087..27.087 rows=0 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.002 rows=0 loops=1)
        -> Aggregate using temporary table  (actual time=27.071..27.071 rows=0 loops=1)
            -> Nested loop inner join  (cost=7534.46 rows=5944) (actual time=27.039..27.039 rows=0 loops=1)
                -> Filter: (CourseStats.Professor like 'name')  (cost=5454.15 rows=5944) (actual time=27.037..27.037 rows=0 loops=1)
                    -> Table scan on CourseStats  (cost=5454.15 rows=53499) (actual time=0.031..22.605 rows=53778 loops=1)
                -> Single-row index lookup on CourseProperties using PRIMARY (CourseID=CourseStats.CourseID)  (cost=0.25 rows=1) (never executed)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------+
1 row in set (0.03 sec)
```

After creating an index on the professor column in CourseStats:

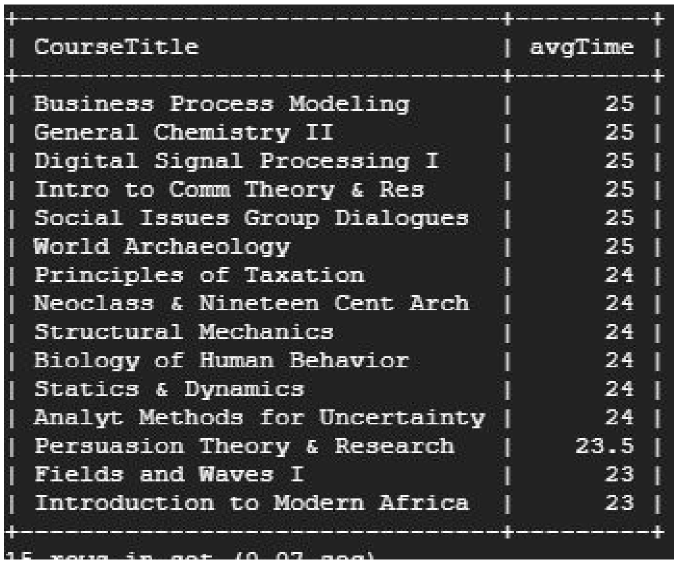


```
| EXPLAIN


+--------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
| -> Sort: <temporary>.numCourses DESC  (actual time=0.045..0.045 rows=0 loops=1)
    -> Stream results  (actual time=0.038..0.038 rows=0 loops=1)
        -> Group aggregate: count(CourseProperties.CourseID)  (actual time=0.037..0.037 rows=0 loops=1)
            -> Nested loop inner join  (cost=1.06 rows=1) (actual time=0.036..0.036 rows=0 loops=1)
                -> Index range scan on CourseStats using idx_professor, with index condition: (CourseStats.Professor like 'name')  (cost=0.71 rows=1) (actual time=0.035..0.035 rows=0 loops=1)
                -> Single-row index lookup on CourseProperties using PRIMARY (CourseID=CourseStats.CourseID)  (cost=0.35 rows=1) (never executed)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```



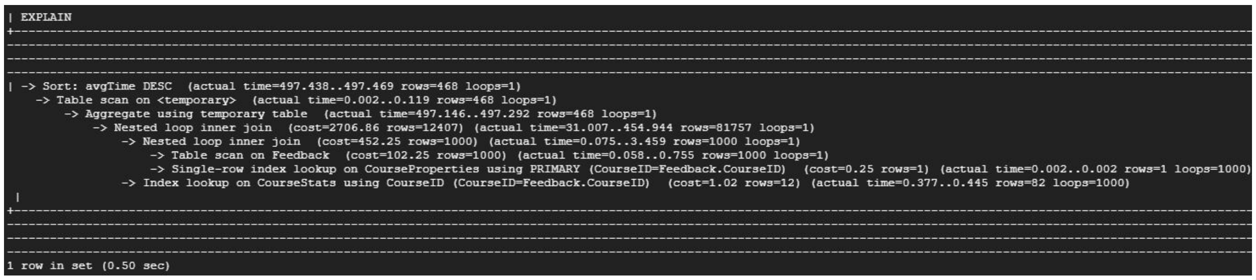
Conclusion: Creating an index on professor improved the performance to nearly 0 seconds execution time. This wasn't a huge performance gain however, as the datasets this query pulls from are pretty small. Over time, as we add more data it might slow down a little, but the tables are not expected to grow significantly.

3) SELECT CourseTitle, Avg(TimeConsumption) as avgTime FROM CourseStats JOIN Feedback USING(CourseID) JOIN CourseProperties USING (CourseId) GROUP BY CourseID ORDER BY avgTime DESC



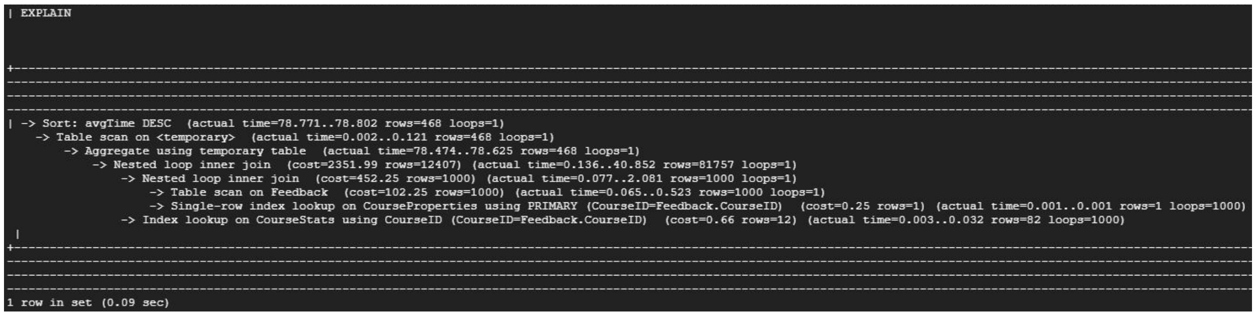
```
+--------------------------------+---------+
| CourseTitle                    | avgTime |
+--------------------------------+---------+
| Business Process Modeling      |      25 |
| General Chemistry II           |      25 |
| Digital Signal Processing I    |      25 |
| Intro to Comm Theory & Res     |      25 |
| Social Issues Group Dialogues  |      25 |
| World Archaeology              |      25 |
| Principles of Taxation         |      24 |
| Neoclass & Nineteen Cent Arch  |      24 |
| Structural Mechanics           |      24 |
| Biology of Human Behavior      |      24 |
| Statics & Dynamics             |      24 |
| Analyt Methods for Uncertainty |      24 |
| Persuasion Theory & Research   |    23.5 |
| Fields and Waves I             |      23 |
| Introduction to Modern Africa  |      23 |
+--------------------------------+---------+
15 rows in set (0.07 sec)
```

Without any indices this query ran in 0.50 seconds:



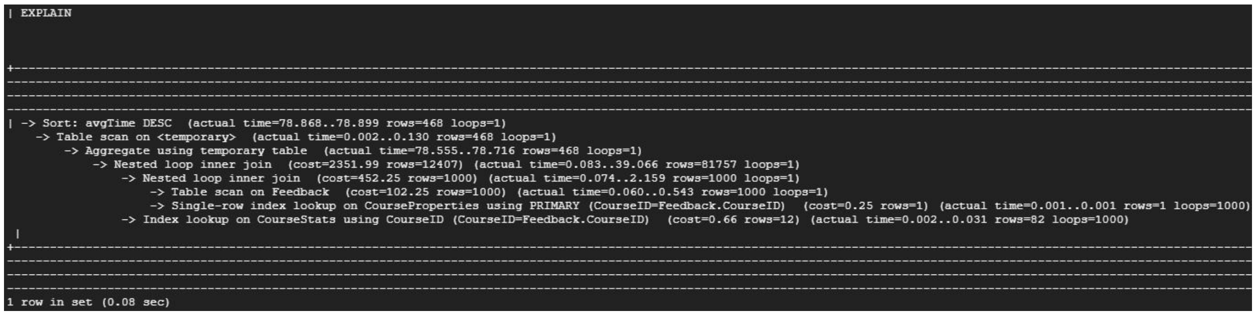
```
| EXPLAIN
+--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
| -> Sort: avgTime DESC  (actual time=497.438..497.469 rows=468 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.119 rows=468 loops=1)
        -> Aggregate using temporary table  (actual time=497.146..497.292 rows=468 loops=1)
            -> Nested loop inner join  (cost=2706.86 rows=12407) (actual time=31.007..454.944 rows=81757 loops=1)
                -> Nested loop inner join  (cost=452.25 rows=1000) (actual time=0.075..3.459 rows=1000 loops=1)
                    -> Table scan on Feedback  (cost=102.25 rows=1000) (actual time=0.058..0.755 rows=1000 loops=1)
                    -> Single-row index lookup on CourseProperties using PRIMARY (CourseID=Feedback.CourseID)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1000)
                -> Index lookup on CourseStats using CourseID (CourseID=Feedback.CourseID)  (cost=1.02 rows=12) (actual time=0.377..0.445 rows=82 loops=1000)
 |
+--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
1 row in set (0.50 sec)
```

With an index on professor in CourseStats:



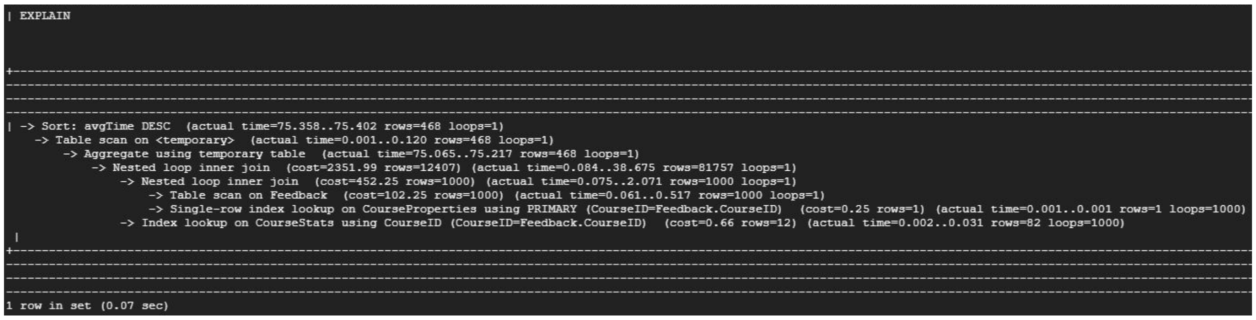
```
| EXPLAIN



+--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
| -> Sort: avgTime DESC  (actual time=78.771..78.802 rows=468 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.121 rows=468 loops=1)
        -> Aggregate using temporary table  (actual time=78.474..78.625 rows=468 loops=1)
            -> Nested loop inner join  (cost=2351.99 rows=12407) (actual time=0.136..40.852 rows=81757 loops=1)
                -> Nested loop inner join  (cost=452.25 rows=1000) (actual time=0.077..2.081 rows=1000 loops=1)
                    -> Table scan on Feedback  (cost=102.25 rows=1000) (actual time=0.065..0.523 rows=1000 loops=1)
                    -> Single-row index lookup on CourseProperties using PRIMARY (CourseID=Feedback.CourseID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
                -> Index lookup on CourseStats using CourseID (CourseID=Feedback.CourseID)  (cost=0.66 rows=12) (actual time=0.003..0.032 rows=82 loops=1000)
 |
+--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
1 row in set (0.09 sec)
```

With an index on course title in CourseProperties:

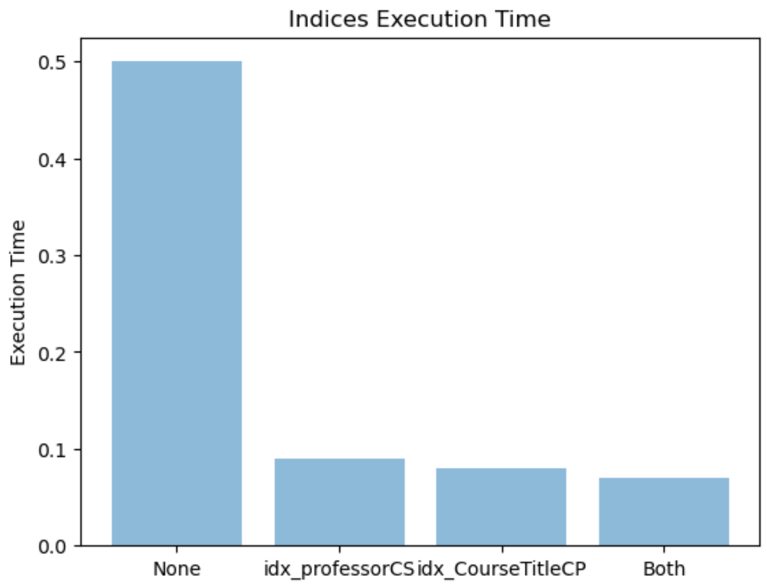


```
| EXPLAIN



+--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
| -> Sort: avgTime DESC  (actual time=78.868..78.899 rows=468 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..0.130 rows=468 loops=1)
        -> Aggregate using temporary table  (actual time=78.555..78.716 rows=468 loops=1)
            -> Nested loop inner join  (cost=2351.99 rows=12407) (actual time=0.083..39.066 rows=81757 loops=1)
                -> Nested loop inner join  (cost=452.25 rows=1000) (actual time=0.074..2.159 rows=1000 loops=1)
                    -> Table scan on Feedback  (cost=102.25 rows=1000) (actual time=0.060..0.543 rows=1000 loops=1)
                    -> Single-row index lookup on CourseProperties using PRIMARY (CourseID=Feedback.CourseID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
                -> Index lookup on CourseStats using CourseID (CourseID=Feedback.CourseID)  (cost=0.66 rows=12) (actual time=0.002..0.031 rows=82 loops=1000)
 |
+--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
1 row in set (0.08 sec)
```

With both indices:

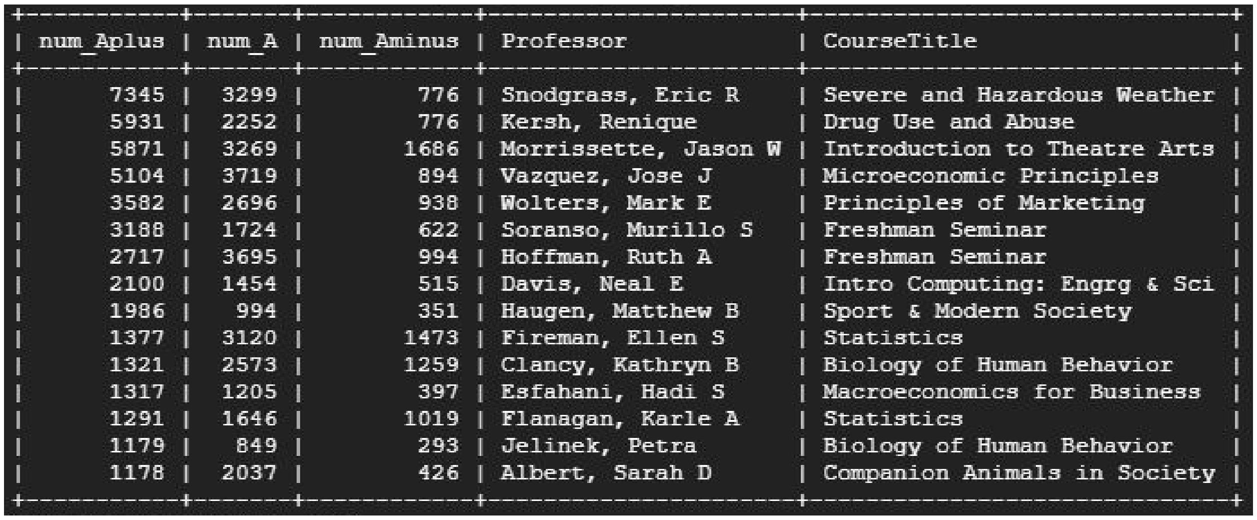


```
| EXPLAIN



+--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
| -> Sort: avgTime DESC  (actual time=75.358..75.402 rows=468 loops=1)
    -> Table scan on <temporary>  (actual time=0.001..0.120 rows=468 loops=1)
        -> Aggregate using temporary table  (actual time=75.065..75.217 rows=468 loops=1)
            -> Nested loop inner join  (cost=2351.99 rows=12407) (actual time=0.084..38.675 rows=81757 loops=1)
                -> Nested loop inner join  (cost=452.25 rows=1000) (actual time=0.075..2.071 rows=1000 loops=1)
                    -> Table scan on Feedback  (cost=102.25 rows=1000) (actual time=0.061..0.517 rows=1000 loops=1)
                    -> Single-row index lookup on CourseProperties using PRIMARY (CourseID=Feedback.CourseID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1000)
                -> Index lookup on CourseStats using CourseID (CourseID=Feedback.CourseID)  (cost=0.66 rows=12) (actual time=0.002..0.031 rows=82 loops=1000)
 |
+--------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------------------------
1 row in set (0.07 sec)
```

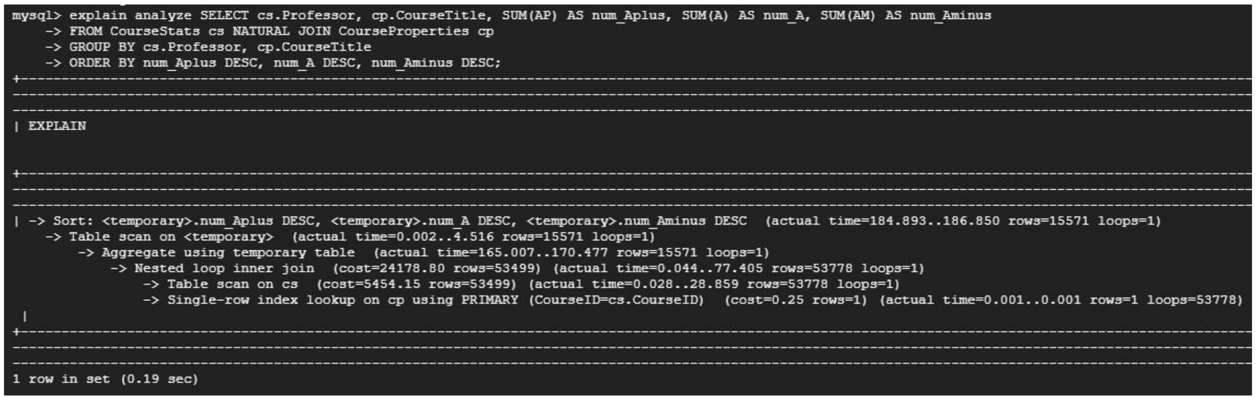


Conclusion: Having an index significantly sped up the query time. There wasn't a significant difference in speedup between two different indices, or even including both. But overall, having an index gave about a 5x speed up from having no index.

4) SELECT cs.Professor, cp.CourseTitle, SUM(AP) AS num_Aplus, SUM(A) AS num_A, SUM(AM) AS num_Aminus FROM CourseStats cs NATURAL JOIN CourseProperties cp GROUP BY cs.Professor, cp.CourseTitle ORDER BY num_Aplus DESC, num_A DESC, num_Aminus DESC;

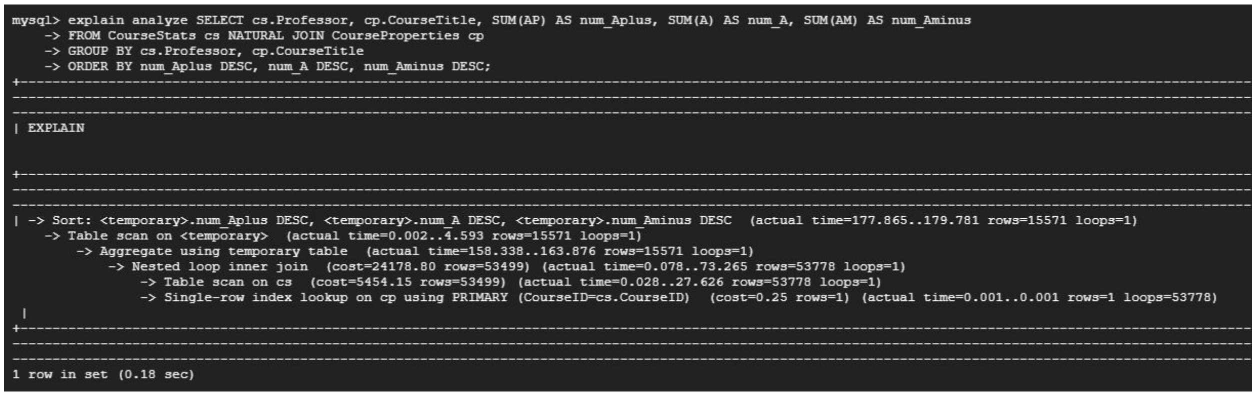

```
+-----------+-------+------------+-----------------------+------------------------------+
| num_Aplus | num_A | num_Aminus | Professor             | CourseTitle                  |
+-----------+-------+------------+-----------------------+------------------------------+
|      7345 |  3299 |        776 | Snodgrass, Eric R     | Severe and Hazardous Weather |
|      5931 |  2252 |        776 | Kersh, Renique        | Drug Use and Abuse           |
|      5871 |  3269 |       1686 | Morrissette, Jason W  | Introduction to Theatre Arts |
|      5104 |  3719 |        894 | Vazquez, Jose J       | Microeconomic Principles     |
|      3582 |  2696 |        938 | Wolters, Mark E       | Principles of Marketing      |
|      3188 |  1724 |        622 | Soranso, Murillo S    | Freshman Seminar             |
|      2717 |  3695 |        994 | Hoffman, Ruth A       | Freshman Seminar             |
|      2100 |  1454 |        515 | Davis, Neal E         | Intro Computing: Engrg & Sci |
|      1986 |   994 |        351 | Haugen, Matthew B     | Sport & Modern Society       |
|      1377 |  3120 |       1473 | Fireman, Ellen S      | Statistics                   |
|      1321 |  2573 |       1259 | Clancy, Kathryn B     | Biology of Human Behavior    |
|      1317 |  1205 |        397 | Esfahani, Hadi S      | Macroeconomics for Business  |
|      1291 |  1646 |       1019 | Flanagan, Karle A     | Statistics                   |
|      1179 |   849 |        293 | Jelinek, Petra        | Biology of Human Behavior    |
|      1178 |  2037 |        426 | Albert, Sarah D       | Companion Animals in Society |
+-----------+-------+------------+-----------------------+------------------------------+
```

Without any indices this query ran in 0.19 seconds:



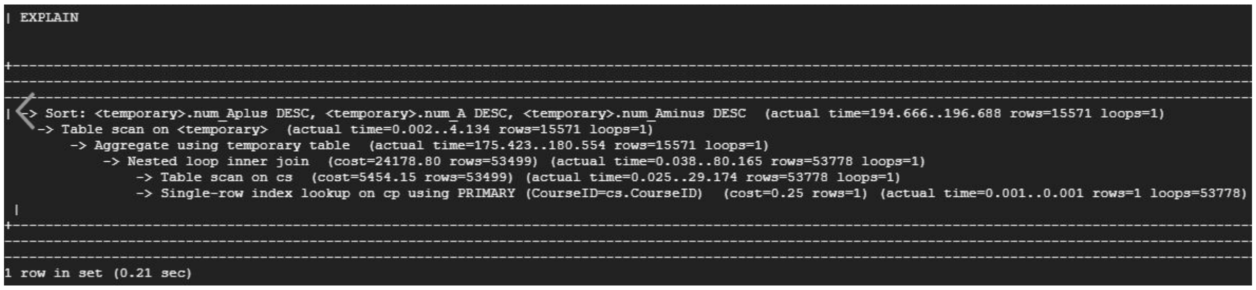
```
mysql> explain analyze SELECT cs.Professor, cp.CourseTitle, SUM(AP) AS num_Aplus, SUM(A) AS num_A, SUM(AM) AS num_Aminus
    -> FROM CourseStats cs NATURAL JOIN CourseProperties cp
    -> GROUP BY cs.Professor, cp.CourseTitle
    -> ORDER BY num_Aplus DESC, num_A DESC, num_Aminus DESC;
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
| -> Sort: <temporary>.num_Aplus DESC, <temporary>.num_A DESC, <temporary>.num_Aminus DESC  (actual time=184.893..186.850 rows=15571 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..4.516 rows=15571 loops=1)
        -> Aggregate using temporary table  (actual time=165.007..170.477 rows=15571 loops=1)
            -> Nested loop inner join  (cost=24178.80 rows=53499) (actual time=0.044..77.405 rows=53778 loops=1)
                -> Table scan on cs  (cost=5454.15 rows=53499) (actual time=0.028..28.859 rows=53778 loops=1)
                -> Single-row index lookup on cp using PRIMARY (CourseID=cs.CourseID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=53778)
 |
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.19 sec)
```

With an index on professor in CourseStats:



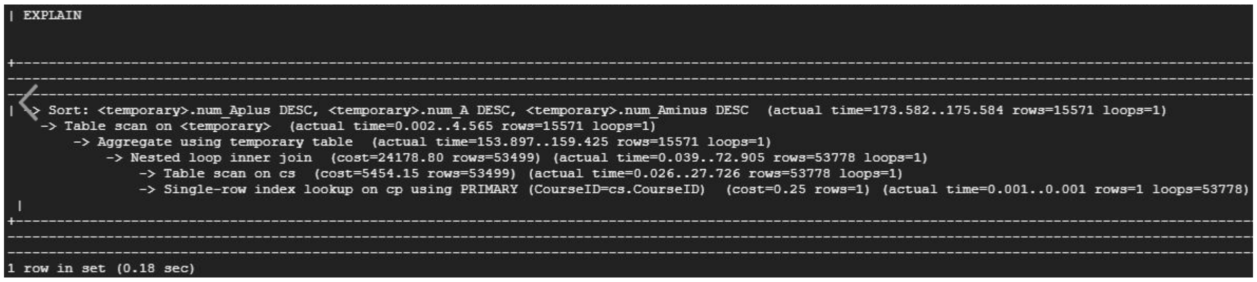
```
mysql> explain analyze SELECT cs.Professor, cp.CourseTitle, SUM(AP) AS num_Aplus, SUM(A) AS num_A, SUM(AM) AS num_Aminus
    -> FROM CourseStats cs NATURAL JOIN CourseProperties cp
    -> GROUP BY cs.Professor, cp.CourseTitle
    -> ORDER BY num_Aplus DESC, num_A DESC, num_Aminus DESC;
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
| -> Sort: <temporary>.num_Aplus DESC, <temporary>.num_A DESC, <temporary>.num_Aminus DESC  (actual time=177.865..179.781 rows=15571 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..4.593 rows=15571 loops=1)
        -> Aggregate using temporary table  (actual time=158.338..163.876 rows=15571 loops=1)
            -> Nested loop inner join  (cost=24178.80 rows=53499) (actual time=0.078..73.265 rows=53778 loops=1)
                -> Table scan on cs  (cost=5454.15 rows=53499) (actual time=0.028..27.626 rows=53778 loops=1)
                -> Single-row index lookup on cp using PRIMARY (CourseID=cs.CourseID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=53778)
 |
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.18 sec)
```

With an index on course title in CourseProperties:

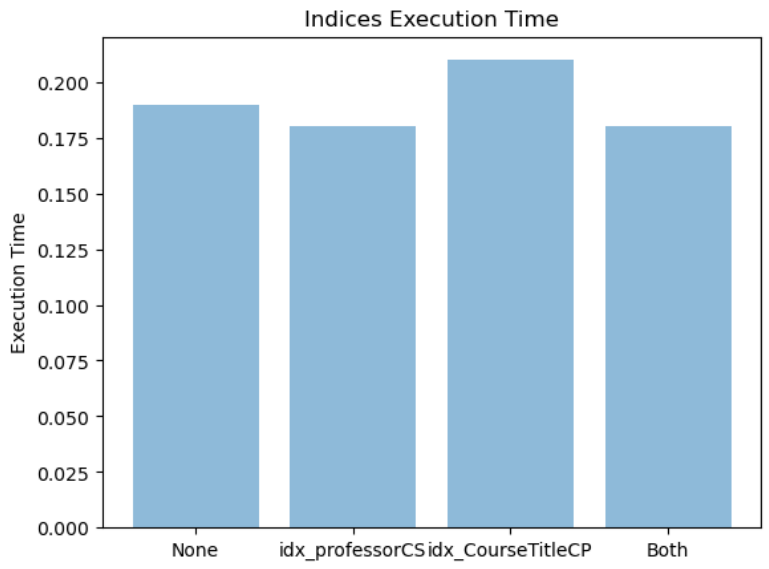


```
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
| -> Sort: <temporary>.num_Aplus DESC, <temporary>.num_A DESC, <temporary>.num_Aminus DESC  (actual time=194.666..196.688 rows=15571 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..4.134 rows=15571 loops=1)
        -> Aggregate using temporary table  (actual time=175.423..180.554 rows=15571 loops=1)
            -> Nested loop inner join  (cost=24178.80 rows=53499) (actual time=0.038..80.165 rows=53778 loops=1)
                -> Table scan on cs  (cost=5454.15 rows=53499) (actual time=0.025..29.174 rows=53778 loops=1)
                -> Single-row index lookup on cp using PRIMARY (CourseID=cs.CourseID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=53778)
 |
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.21 sec)
```

After both indices:



```
| EXPLAIN


+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
| -> Sort: <temporary>.num_Aplus DESC, <temporary>.num_A DESC, <temporary>.num_Aminus DESC  (actual time=173.582..175.584 rows=15571 loops=1)
    -> Table scan on <temporary>  (actual time=0.002..4.565 rows=15571 loops=1)
        -> Aggregate using temporary table  (actual time=153.897..159.425 rows=15571 loops=1)
            -> Nested loop inner join  (cost=24178.80 rows=53499) (actual time=0.039..72.905 rows=53778 loops=1)
                -> Table scan on cs  (cost=5454.15 rows=53499) (actual time=0.026..27.726 rows=53778 loops=1)
                -> Single-row index lookup on cp using PRIMARY (CourseID=cs.CourseID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=53778)
 |
+------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.18 sec)
```



Conclusion: After creating indices on the Professor and Courses table respectively, the individual performances seemed to hover around 0.19 seconds. However, after I combined both indices, the query ran in 0.18 seconds which is 0.01 seconds faster than the original query (without indices). Therefore, I decided to combine the idx_professor and idx_course indices to have a quicker query.

G. Where We Collected Data and How

Our data comes from these sources:

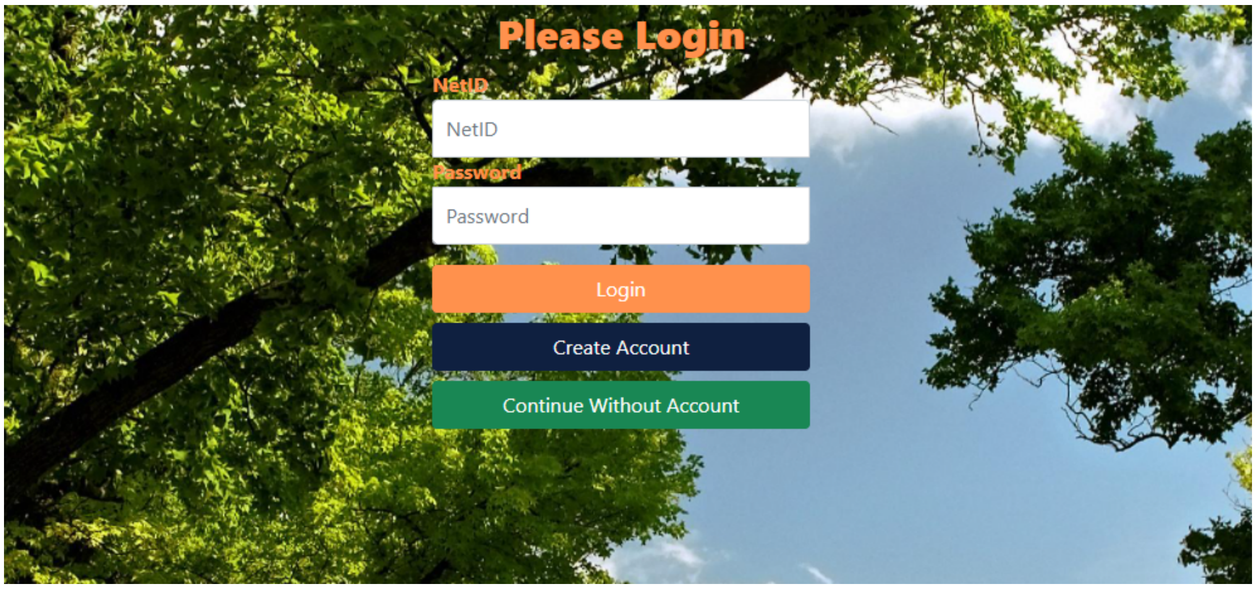
GPA: https://github.com/wadefagen/datasets/tree/master/gpa

COURSES: https://courses.illinois.edu/cisdocs/explorer

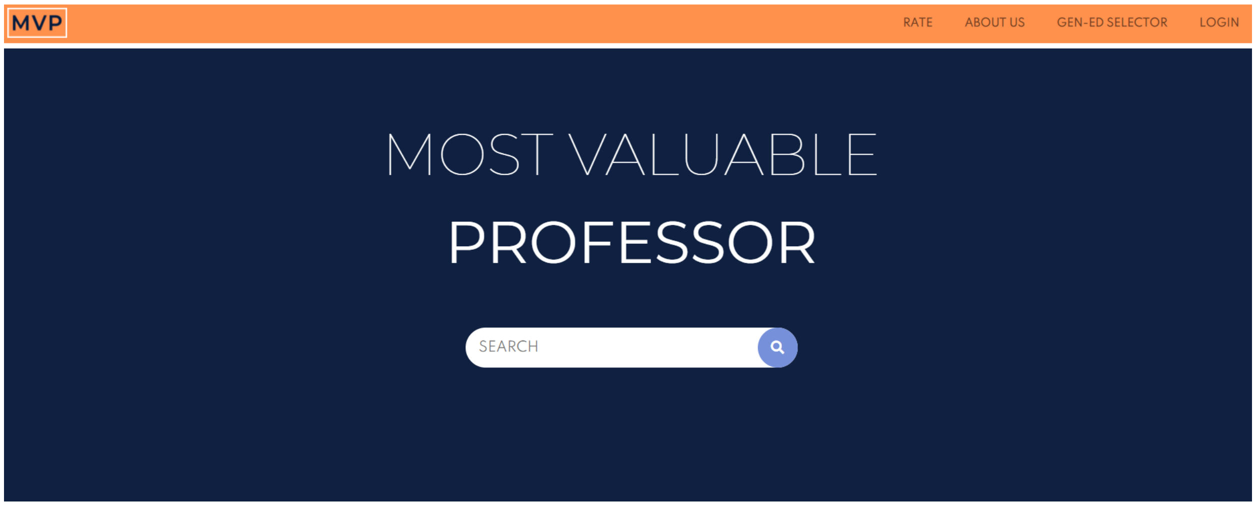
The GPA data was processed with PANDAS, and the course property data was processed with requests to the course explorer API. The profanity data was gathered online. All other data was randomly generated (we plan to delete all the randomly generated data).

H. Application Design and Features

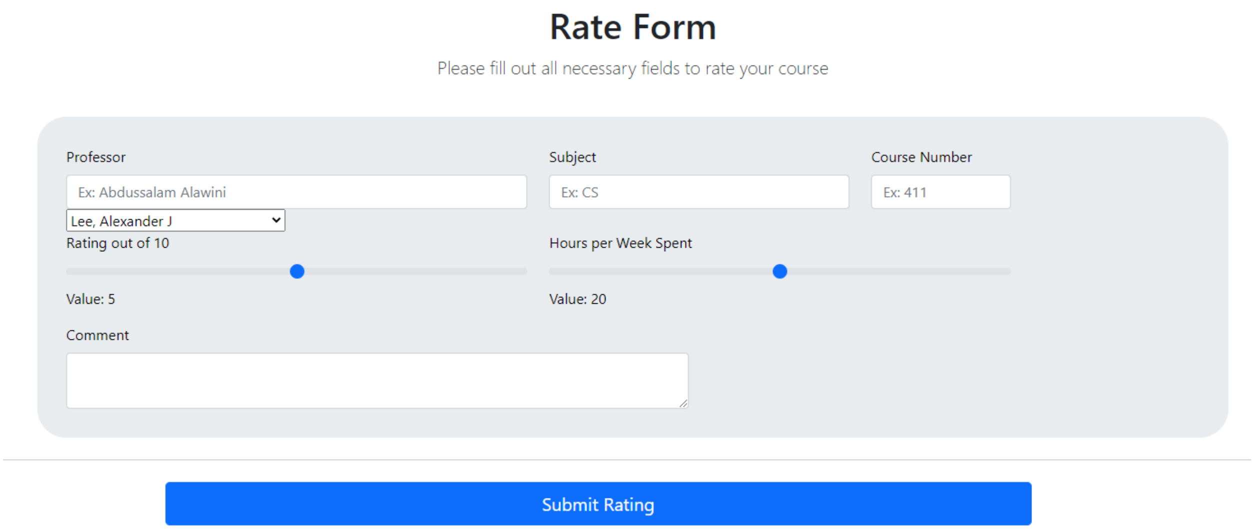
Login Page: Where users can login, create an account or chose to continue without an account (although this prevents this from leaving a rating)



Main Page / Search Page: Users are directed here first and after/if they chose to login. This page has a navigation bar at the top where users can navigate all other parts of the website including the Rate page, About Us page, Gen-Ed Selector, and Login/Logout Page. The main search bar can be used to search for courses, Professors, or other a combination of the two.



Rate: Here a user can submit their rating to a professor/course.


Rate Form

Please fill out all necessary fields to rate your course

Professor: Ex: Abdussalam Alawini

Lee, Alexander J

Subject: Ex: CS

Course Number: Ex: 411

Rating out of 10

Value: 5

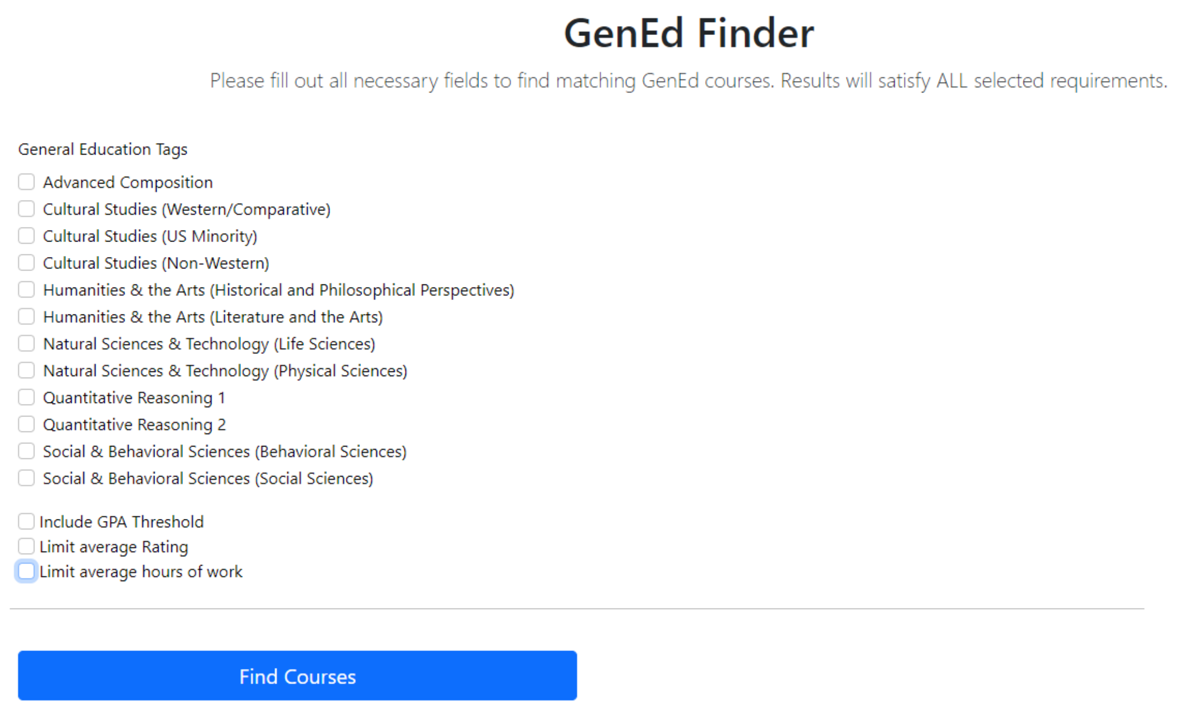
Hours per Week Spent

Value: 20

Comment

About Us: This page displays a description and contact information for the creators (us).

Gen-Ed Selector: This page allows a user to search for gen-eds using several different filters. This is also our implementation of a stored procedure.



GenEd Finder

Please fill out all necessary fields to find matching GenEd courses. Results will satisfy ALL selected requirements.

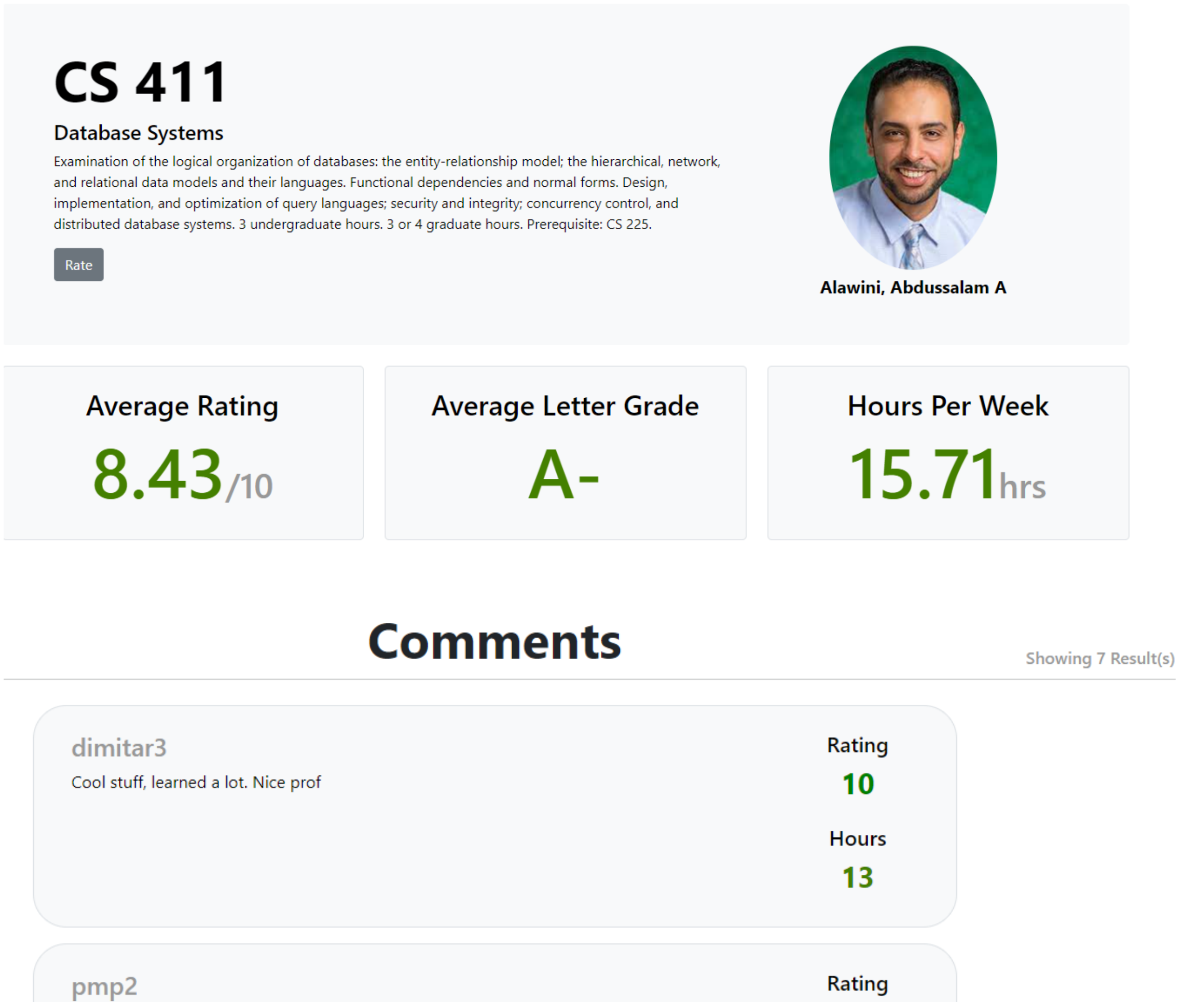
General Education Tags

- [ ] Advanced Composition
- [ ] Cultural Studies (Western/Comparative)
- [ ] Cultural Studies (US Minority)
- [ ] Cultural Studies (Non-Western)
- [ ] Humanities & the Arts (Historical and Philosophical Perspectives)
- [ ] Humanities & the Arts (Literature and the Arts)
- [ ] Natural Sciences & Technology (Life Sciences)
- [ ] Natural Sciences & Technology (Physical Sciences)
- [ ] Quantitative Reasoning 1
- [ ] Quantitative Reasoning 2
- [ ] Social & Behavioral Sciences (Behavioral Sciences)
- [ ] Social & Behavioral Sciences (Social Sciences)

- [ ] Include GPA Threshold
- [ ] Limit average Rating
- [ ] Limit average hours of work

Admin Pages: This part of our website is only for the admin users, where all the tables can be accessed and basic CRUD commands can be performed on them along with a simple search functionality.

Professor/Course page: This is the display page for the professor/course combo that the user selects (after finding it through main search or gen-ed selector). This contains the most important information a user would want including average rating, average letter grade/GPA, average self-reported hours spent per week, along with some data/statistic visualizations and the comment section. There is also the course description along with any prerequisites and the credit hours and the professor name and picture. Finally, there is a button to take you directly to the rating page (with the information automatically filled out for the current professor being observed).



CS 411

Database Systems

Examination of the logical organization of databases: the entity-relationship model; the hierarchical, network, and relational data models and their languages. Functional dependencies and normal forms. Design, implementation, and optimization of query languages; security and integrity; concurrency control, and distributed database systems. 3 undergraduate hours. 3 or 4 graduate hours. Prerequisite: CS 225.

Rate

Alawini, Abdussalam A

Average Rating	Average Letter Grade	Hours Per Week
8.43/10	A-	15.71hrs

Comments

Showing 7 Result(s)

dimitar3

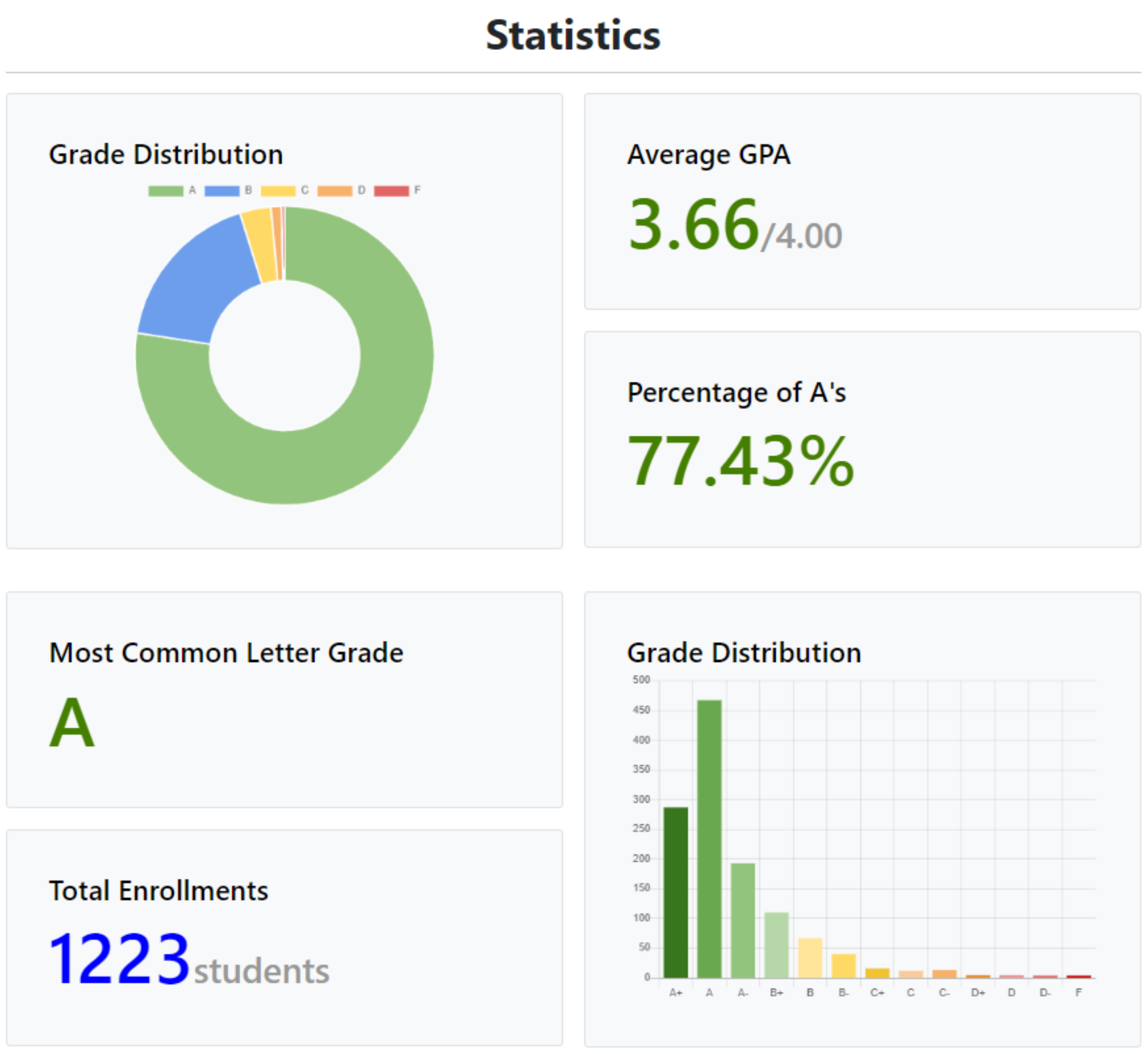
Cool stuff, learned a lot. Nice prof

Rating
10

Hours
13

pmp2

Rating


Statistics

Grade Distribution

Average GPA
3.66/4.00

Percentage of A's
77.43%

Most Common Letter Grade
A

Grade Distribution

Total Enrollments
1223students

I. Advanced Database Program (Stored procedure + Trigger)

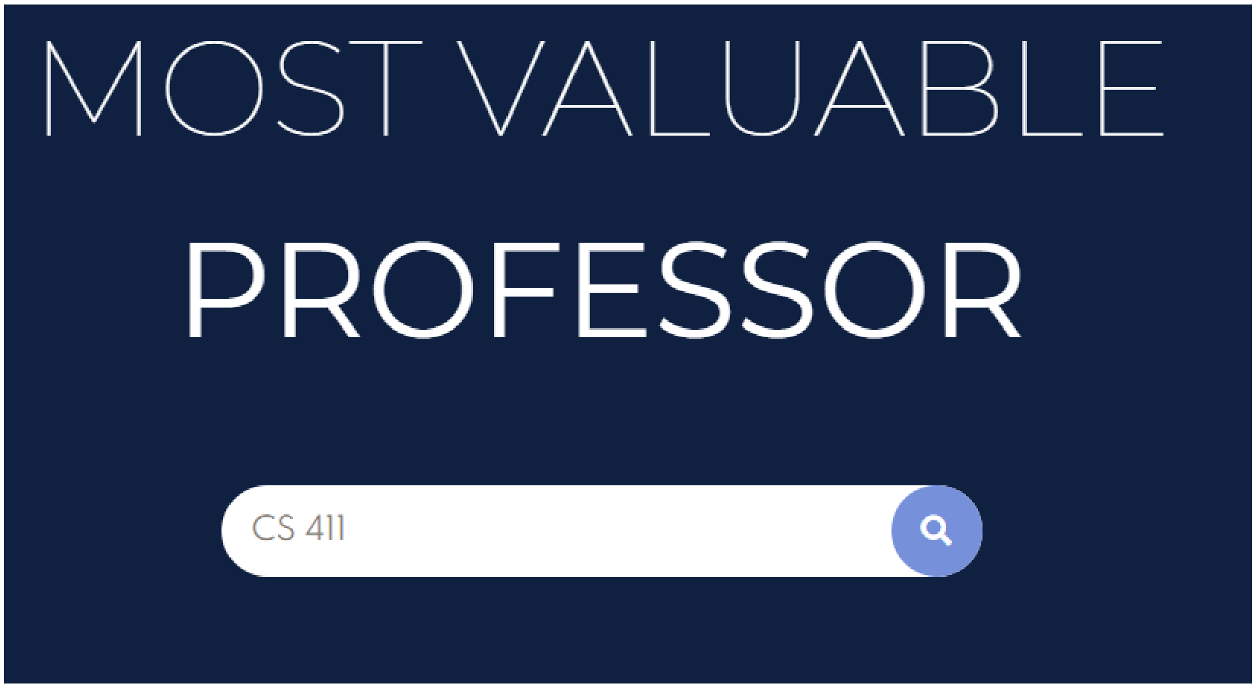
Our database incorporated a trigger (before insert/update) that would check feedback entries for profanity. This is useful to our application because we don't want users to be able to leave comments with profanity on our website.

Our database also included a stored procedure that was designed to help users search the GenEds here at UIUC. Because we wanted to give the users lots of options in refining their search, the conditional capacities of a stored procedure were appropriate. Additionally, this procedure uses an intermediate table to help process the results, which further exemplifies the choice to use a stored procedure. Essentially, users would be able to choose which GenEd tags they would like the results to include, and then they could choose an average GPA, rating, and/or work hours threshold. They were also given the option to choose which years they wanted the results to consider. If a user wanted only more recent results, they could specify so. They could also choose to exclude covid semesters from their results, as those instances of a course could significantly deviate from the in-person instances.

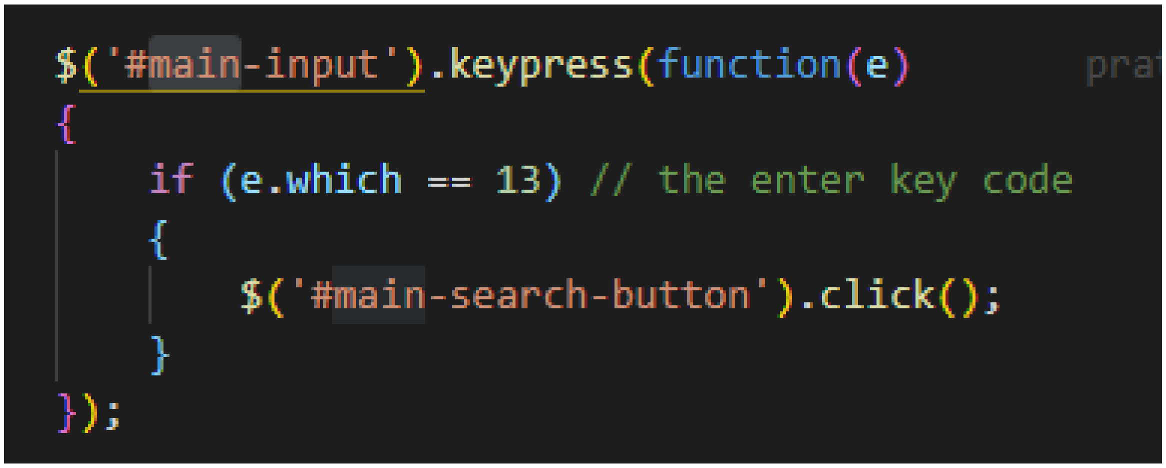
J. Dataflow

Dataflow for the main search page:

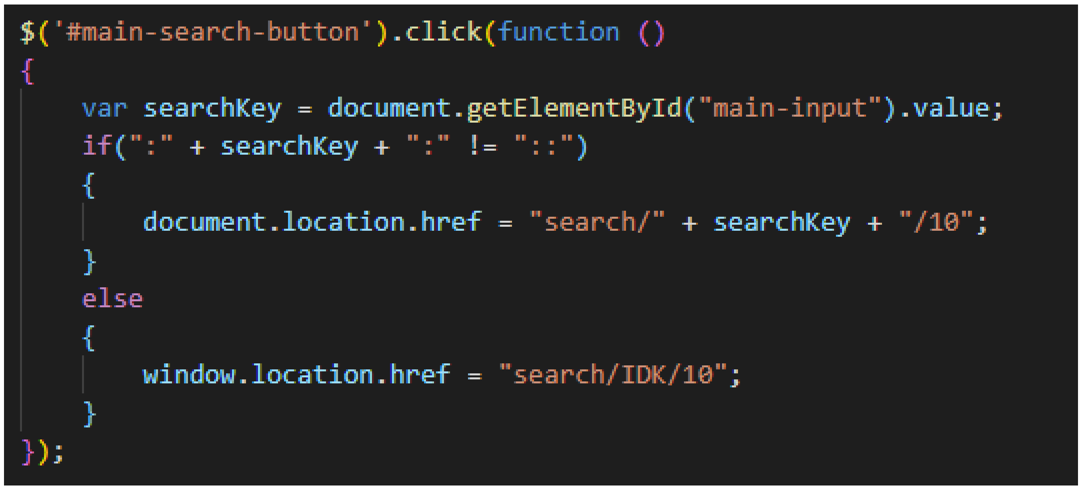
1) User enters text into search field:



2) User presses enter or clicks the search button, javascript redirects to a new route

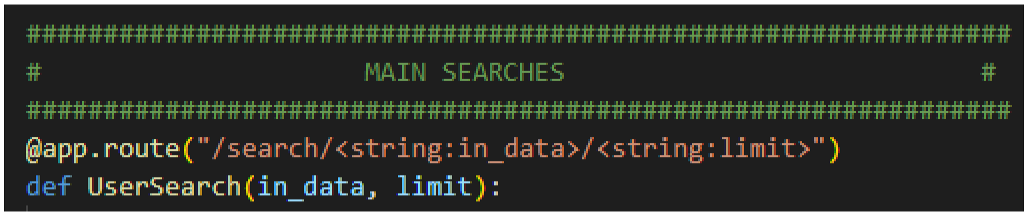


```
$('#main-input').keypress(function(e)          pra
{
    if (e.which == 13) // the enter key code
    {
        $('#main-search-button').click();
    }
});
```



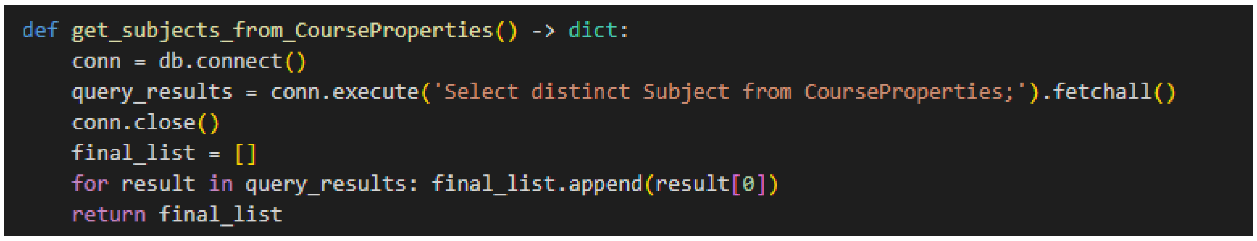
```
$('#main-search-button').click(function ()
{
    var searchKey = document.getElementById("main-input").value;
    if(":" + searchKey + ":" != "::")
    {
        document.location.href = "search/" + searchKey + "/10";
    }
    else
    {
        window.location.href = "search/IDK/10";
    }
});
```

3) This gets passed into the routes.py calling a specific function



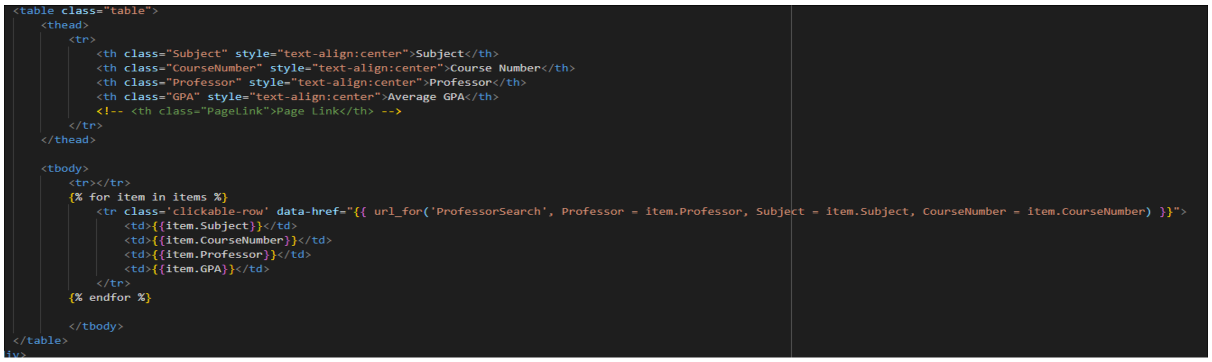
```
################################################################
#                     MAIN SEARCHES                          #
################################################################
@app.route("/search/<string:in_data>/<string:limit>")
def UserSearch(in_data, limit):
```

4) Which calls helper functions in database.py (interacting with the database)



```python
def get_subjects_from_CourseProperties() -> dict:
    conn = db.connect()
    query_results = conn.execute('Select distinct Subject from CourseProperties;').fetchall()
    conn.close()
    final_list = []
    for result in query_results: final_list.append(result[0])
    return final_list
```

5) And finally, routes.py finishes by rendering a html template to the front end



```html
<table class="table">
    <thead>
        <tr>
            <th class="Subject" style="text-align:center">Subject</th>
            <th class="CourseNumber" style="text-align:center">Course Number</th>
            <th class="Professor" style="text-align:center">Professor</th>
            <th class="GPA" style="text-align:center">Average GPA</th>
            <!-- <th class="PageLink">Page Link</th> -->
        </tr>
    </thead>

    <tbody>
        <tr></tr>
        {% for item in items %}
            <tr class='clickable-row' data-href="{{ url_for('ProfessorSearch', Professor = item.Professor, Subject = item.Subject, CourseNumber = item.CourseNumber) }}">
                <td>{{item.Subject}}</td>
                <td>{{item.CourseNumber}}</td>
                <td>{{item.Professor}}</td>
                <td>{{item.GPA}}</td>
            </tr>
        {% endfor %}

        </tbody>
</table>
iv>
```

This process is very similar across all functions of our website.

K. One Technical Challenge Encountered

A challenge we faced, specifically the day before our final demo, was hosting our project with GCP. Our stored procedure feature did not work on our deployed site, and we figured out that the static folder (specifically a javascript file) was not being updated even after redeployment. We cleared the cache, tried to mess with settings, tried setting the update rate to a faster one, deleted buckets and redeployed but nothing worked. It simply needed time in order to update those static files. Thankfully, everything was updated by the demo the next day, but it was worrying. Our advice would be to finish the final demo early and host as early as possible (at least a couple days before) in order for everything to be updated before presenting. Alternatively, you could wait to upload hosting files until everything is sufficiently tested and thus there would be no need to update any files.

L. Did Everything Go As Planned?

No, there were some details in our implementations that had to be changed. For instance, our initial idea was to have students use the university's login system in order to handle verification easier, however this proved a little more difficult and we did not have enough time to properly implement this. Additionally, we did not foresee a need for admin users, but we added this toward the end of development, resulting in a suboptimal database design as we could have had a single registered user table but each user could have a specified role. Another thing we had to change toward the end of our development was to include a table for professor images, this was a detail we forgot to implement.

M. Division of Labor

Pratik - Backend, Database implementation

Michael - Backend, Database implementation (as well as stored procedure + trigger)

Dimitar - Backend, Frontend

Omar - Frontend, Design

We managed our teamwork well, we met every week on saturday at a specified time and we all managed to do our part. This was all of our first time with web development but we learned and struggled together.
