Database Implementation and Indexing

For the implementation of our HelpMeGraduate Website, we created our database on GCP (Google Cloud Platform)

```
yayitsthunder@cloudshell:~ (team075-phantom-troupe)$ gcloud sql connect help-me-graduate --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 20413
Server version: 8.0.26-google (Google)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

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

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

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| helpmegraduate     |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

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

Database changed
mysql> show tables;
+--------------------------+
| Tables_in_helpmegraduate |
+--------------------------+
| Courses                  |
| CoursesHasGrades         |
| Difficulty               |
| DifficultyOfCourses      |
| Grades                   |
| Professors               |
| Search                   |
| Teach                    |
| UserInfo                 |
+--------------------------+
9 rows in set (0.00 sec)
```

Figure 1. Connection to GCP

Data Definition Language (DDL) commands for our main tables (5 main tables and 4 relationship tables) (attached .sql file in the sql directory)

```
CREATE TABLE Search(
CRN INT references Courses(CRN) ON DELETE SET NULL,
UserID INT references UserInfo(UserID) ON DELETE SET NULL,
PRIMARY KEY (CRN,UserID));

CREATE TABLE DifficultyOfCourses(
CRN INT references Courses(CRN) ON DELETE SET NULL,
DifficultyLevel INT references Difficulty(DifficultyLevel) ON DELETE SET NULL,
PRIMARY KEY (CRN,DifficultyLevel));

CREATE TABLE CoursesHasGrades(
CRN INT references Courses(CRN) ON DELETE SET NULL,
GradeLevel DECIMAL(6,2) references Grades(GradeLevel) ON DELETE SET NULL,
PRIMARY KEY (CRN,GradeLevel));

CREATE TABLE Teach(
CRN INT references Courses(CRN) ON DELETE SET NULL,
ProfessorID INT references Professors(ProfessorID) ON DELETE SET NULL,
PRIMARY KEY (CRN,ProfessorID));
```

```
CREATE TABLE UserInfo(
UserID INT NOT NULL,
UserFirstName VARCHAR(50),
UserLastName VARCHAR(50),
PRIMARY KEY (UserID));

CREATE TABLE Courses(
CRN INT NOT NULL,
Year_Term VARCHAR(15),
Department VARCHAR(15),
CourseNumber VARCHAR(10),
CourseName VARCHAR(100),
PRIMARY KEY (CRN));

CREATE TABLE Professors(
ProfessorID INT NOT NULL,
ProfessorName VARCHAR(50),
Rate INT,
Deparment VARCHAR(15),
PRIMARY KEY (ProfessorID));

CREATE TABLE Grades(
GradeLevel DECIMAL(6,2) NOT NULL,
AvgGPA DECIMAL(3,2),
A INT,
B INT,
C INT,
D INT,
F INT,
W INT,
PRIMARY KEY (GradeLevel));

CREATE TABLE Difficulty(
DifficultyLevel INT,
Exams INT,
Discussions INT,
Labs_Mps INT,
Homework INT,
PRIMARY KEY (DifficultyLevel));
```

Figure 2. Create Tables

Data in our main tables (since we haven't developed our website yet, there's no data in UserInfo table)

```
mysql> select COUNT(*) FROM Courses;
+----------+
| COUNT(*) |
+----------+
|     5366 |
+----------+
1 row in set (0.00 sec)

mysql> select COUNT(*) FROM Professors;
+----------+
| COUNT(*) |
+----------+
|     3152 |
+----------+
1 row in set (0.01 sec)

mysql> select COUNT(*) FROM Difficulty;
+----------+
| COUNT(*) |
+----------+
|     1556 |
+----------+
1 row in set (0.00 sec)

mysql> select COUNT(*) FROM Grades;
+----------+
| COUNT(*) |
+----------+
|     3505 |
+----------+
1 row in set (0.00 sec)
```

Advanced Queries with Analysis of Index

Query 1:

```
-- Calculate the average GPA and average DifficultyLevel of all the ECE course
-- select the ECE course that both have higher GPA and lower DifficultyLevel
-- Also print out the professor that teach the course
-- Order by CourseNumber
SELECT Courses.Department, Courses.CourseNumber, Professors.ProfessorName
FROM Courses JOIN DifficultyOfCourses USING (CRN) JOIN Difficulty USING (DifficultyLevel)
JOIN CoursesHasGrades USING (CRN) JOIN Grades USING (GradeLevel) JOIN Teach USING (CRN) JOIN Professors USING (ProfessorID) JOIN

(SELECT AVG(d.DifficultyLevel) AS AvgDifECE
FROM Courses c JOIN DifficultyOfCourses USING (CRN) JOIN Difficulty d USING (DifficultyLevel)
GROUP BY c.Department
HAVING c.Department='ECE') AvgDif JOIN

(SELECT AVG(g.AvgGPA) AS AvgGPAECE
FROM Courses c1 JOIN CoursesHasGrades USING (CRN) JOIN Grades g USING (GradeLevel)
GROUP BY c1.Department
HAVING c1.Department='ECE') TAvgGPA
WHERE Courses.Department='ECE' AND Grades.AvgGPA > TAvgGPA.AvgGPAECE AND Difficulty.DifficultyLevel < AvgDif.AvgDifECE
ORDER BY Courses.CourseNumber
```

Figure 4. Query1

Output of the first 15 rows:

```
mysql> SELECT Courses.Department, Courses.CourseNumber, Professors.ProfessorName
    -> FROM Courses JOIN DifficultyOfCourses USING (CRN) JOIN Difficulty USING (DifficultyLevel)
    -> JOIN CoursesHasGrades USING (CRN) JOIN Grades USING (GradeLevel) JOIN Teach USING (CRN) JOIN Professors USING (ProfessorID) JOIN
    ->
    -> (SELECT AVG(d.DifficultyLevel) AS AvgDifECE
    -> FROM Courses c JOIN DifficultyOfCourses USING (CRN) JOIN Difficulty d USING (DifficultyLevel)
    -> GROUP BY c.Department
    -> HAVING c.Department='ECE') AvgDif JOIN
    ->
    -> (SELECT AVG(g.AvgGPA) AS AvgGPAECE
    -> FROM Courses c1 JOIN CoursesHasGrades USING (CRN) JOIN Grades g USING (GradeLevel)
    -> GROUP BY c1.Department
    -> HAVING c1.Department='ECE') TAvgGPA
    -> WHERE Courses.Department='ECE' AND Grades.AvgGPA > TAvgGPA.AvgGPAECE AND Difficulty.DifficultyLevel < AvgDif.AvgDifECE
    -> ORDER BY Courses.CourseNumber
    -> LIMIT 15;
+------------+--------------+---------------------------+
| Department | CourseNumber | ProfessorName             |
+------------+--------------+---------------------------+
| ECE        | 206          | Radhakrishnan, Chandrasek |
| ECE        | 298          | Allen, Jont               |
| ECE        | 307          | Gross, George             |
| ECE        | 311          | Katselis, Dimitrios       |
| ECE        | 316          | Hillmer, Philip M         |
| ECE        | 343          | Radhakrishnan, Chandrasek |
| ECE        | 395          | Haken, Lippold            |
| ECE        | 398          | Miller, Andrew E          |
| ECE        | 398          | Do, Minh N                |
| ECE        | 401          | Hasegawa-Johnso, Mark A   |
| ECE        | 402          | Haken, Lippold            |
| ECE        | 417          | Hasegawa-Johnso, Mark A   |
| ECE        | 418          | Moulin, Pierre            |
| ECE        | 420          | Moon, Thomas              |
| ECE        | 435          | Caesar, Matthew C         |
+------------+--------------+---------------------------+
15 rows in set (0.07 sec)
```

Figure 5. Output of the query

Indexing:

Figure 6. Analysis of the query

We can find that without adding the index, the query needs to scan all the rows to filter out the ECE department and calculate the averageGPA or the average difficulty level, so it might be useful to add index to the department

```
mysql> CREATE INDEX department_idx ON Courses(Department)
    -> ;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe Courses;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| CRN          | int          | NO   | PRI | NULL    |       |
| Year_Term    | varchar(15)  | YES  |     | NULL    |       |
| Department   | varchar(15)  | YES  | MUL | NULL    |       |
| CourseNumber | varchar(10)  | YES  |     | NULL    |       |
| CourseName   | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Figure 7. Add Index

```
| -> Limit: 15 row(s)  (actual time=38.214..38.216 rows=15 loops=1)
    -> Sort: Courses.CourseNumber, limit input to 15 row(s) per chunk  (actual time=38.213..38.215 rows=15 loops=1)
        -> Stream results  (cost=25305.53 rows=0) (actual time=38.136..38.182 rows=40 loops=1)
            -> Filter: (Grades.AvgGPA > TAvgGPA.AvgGPAECE)  (cost=25305.53 rows=0) (actual time=38.133..38.163 rows=40 loops=1)
                -> Inner hash join (no condition)  (cost=25305.53 rows=0) (actual time=38.130..38.151 rows=69 loops=1)
                    -> Table scan on TAvgGPA  (cost=2.50..2.50 rows=0) (actual time=0.001..0.001 rows=1 loops=1)
                        -> Materialize  (cost=2.50..2.50 rows=0) (actual time=19.371..19.371 rows=1 loops=1)
                            -> Filter: (c1.Department = 'ECE')  (actual time=19.294..19.345 rows=1 loops=1)
                                -> Table scan on <temporary>  (actual time=0.002..0.035 rows=167 loops=1)
                                    -> Aggregate using temporary table  (actual time=19.285..19.328 rows=167 loops=1)
                                        -> Nested loop inner join  (cost=4295.55 rows=5366) (actual time=0.038..14.683 rows=5366 loops=1)
                                            -> Nested loop inner join  (cost=2417.45 rows=5366) (actual time=0.034..8.537 rows=5366 loops=1)
                                                -> Index scan on CoursesHasGrades using PRIMARY  (cost=539.35 rows=5366) (actual time=0.027..1.336 rows=5366 loops=1)
                                                -> Single-row index lookup on g using PRIMARY (GradeLevel=CoursesHasGrades.GradeLevel)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5366)
                                            -> Single-row index lookup on c1 using PRIMARY (CRN=CoursesHasGrades.CRN)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5366)
```

Figure 8. Analysis of the query (First part)

There do exist some improvement on the total time, almost 2 secs, this is reasonable since the database does not need to scan every row and determine the department, it can find the department of ECE using the index.

Also, I suppose the AvgGPA also matters, since we need to find the AvgGPA that is higher in the whole department (drop Department_idx first)

```
mysql> CREATE INDEX AvgGPA_idx ON Grades(AvgGPA);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe Grades;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| GradeLevel | decimal(6,2) | NO   | PRI | NULL    |       |
| AvgGPA     | decimal(3,2) | YES  | MUL | NULL    |       |
| A          | int          | YES  |     | NULL    |       |
| B          | int          | YES  |     | NULL    |       |
| C          | int          | YES  |     | NULL    |       |
| D          | int          | YES  |     | NULL    |       |
| F          | int          | YES  |     | NULL    |       |
| W          | int          | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
8 rows in set (0.00 sec)
```

Figure 9. Add Index

Figure 10. Analysis of the query after adding GPA index

The cost time also decreases, this is also reasonable since the database do esn't need to find the AvgGPA that is lower than the calculated total AvgG PA.

Now I wonder whether the cost time would increase if we add a trivial inde x. (drop other indexes first)

```
mysql> CREATE INDEX trival_idx ON Courses(CourseNumber);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> describe Courses;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| CRN          | int          | NO   | PRI | NULL    |       |
| Year_Term    | varchar(15)  | YES  |     | NULL    |       |
| Department   | varchar(15)  | YES  |     | NULL    |       |
| CourseNumber | varchar(10)  | YES  | MUL | NULL    |       |
| CourseName   | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Figure 11. Adding a trivial index on Courses

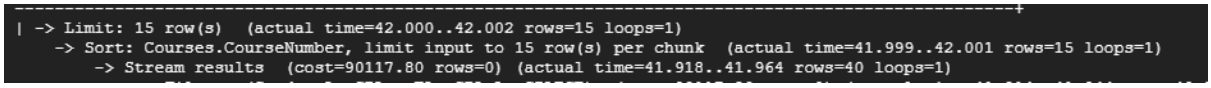

Figure 12. Analysis of the query

We can find that the total time increases, but I think there might be some systematic error when running the query in the database. Anyway, we should avoid creating such trivial index.

Query 2:

```
-- find the CS courses with high rated profs
SELECT DISTINCT Courses.Department, Courses.CourseNumber,Courses.CourseName, Professors.ProfessorName,Professors.Rate
FROM Courses JOIN Teach USING (CRN) JOIN Professors USING (ProfessorID)
WHERE Courses.Department='CS' AND Professors.Rate > (SELECT avg(p.Rate) as AvgProfRate
                                                     FROM Professors p)
ORDER BY Professors.Rate DESC, Courses.CourseNumber ASC
```

Figure 13. Query2

```
mysql> SELECT DISTINCT Courses.Department, Courses.CourseNumber,Courses.CourseName, Professors.ProfessorName,Professors.Rate
    -> FROM Courses JOIN Teach USING (CRN) JOIN Professors USING (ProfessorID)
    -> WHERE Courses.Department='CS' AND Professors.Rate > (SELECT avg(p.Rate) as AvgProfRate
    -> FROM Professors p)
    -> ORDER BY Professors.Rate DESC, Courses.CourseNumber ASC
    -> LIMIT 15;
+------------+--------------+-----------------------------+-------------------+------+
| Department | CourseNumber | CourseName                  | ProfessorName     | Rate |
+------------+--------------+-----------------------------+-------------------+------+
| CS         | 105          | Intro Computing: Non-Tech   | Zilles, Craig     |   10 |
| CS         | 128          | Intro to Computer Science II | Nowak, Michael    |   10 |
| CS         | 199          | Elements of Game Design     | Shaffer, Eric G   |   10 |
| CS         | 199          | Intro to Comp Sci II        | Nowak, Michael    |   10 |
| CS         | 232          | Computer Architecture II    | Zilles, Craig     |   10 |
| CS         | 296          | Honors Course               | Solomon, Brad R   |   10 |
| CS         | 373          | Theory of Computation       | Meseguer, Jose    |   10 |
| CS         | 398          | Applied Cloud Computing     | Brunner, Robert J |   10 |
| CS         | 412          | Introduction to Data Mining | Banerjee, Arindam |   10 |
| CS         | 414          | Multimedia Systems          | Nahrstedt, Klara  |   10 |
| CS         | 419          | Production Computer Graphics | Shaffer, Eric G   |   10 |
| CS         | 461          | Computer Security I         | Ren, Ling         |   10 |
| CS         | 498          | Probability in CS           | Smaragdis, Paris  |   10 |
| CS         | 498          | Cyber Dystopia              | Gunter, Carl      |   10 |
| CS         | 498          | Virtual Reality             | Shaffer, Eric G   |   10 |
+------------+--------------+-----------------------------+-------------------+------+
15 rows in set (0.01 sec)
```

Figure 14. Print first 15 rows of the second advanced query

Indexing:

```
mysql> create index rate_idx ON Professors(Rate);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Figure 15. Command to create index

```
-> Sort: Professors.Rate DESC, Courses.CourseNumber  (actual time=4.050..4.057 rows=66 loops=1)
    -> Table scan on <temporary>  (cost=0.03..4.89 rows=191) (actual time=0.002..0.010 rows=66 loops=1)
        -> Temporary table with deduplication  (cost=951.57..956.43 rows=191) (actual time=4.000..4.012 rows=66 loops=1)
            -> Nested loop inner join  (cost=932.41 rows=191) (actual time=1.105..3.906 rows=66 loops=1)
                -> Nested loop inner join  (cost=731.54 rows=574) (actual time=0.078..2.617 rows=169 loops=1)
                    -> Filter: (Courses.Department = 'CS')  (cost=544.15 rows=520) (actual time=0.066..2.227 rows=169 loops=1)
                        -> Table scan on Courses  (cost=544.15 rows=5199) (actual time=0.059..1.784 rows=5366 loops=1)
                    -> Index lookup on Teach using PRIMARY (CRN=Courses.CRN)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=169)
                -> Filter: (Professors.Rate > (select #2))  (cost=0.25 rows=0) (actual time=0.007..0.007 rows=0 loops=169)
                    -> Single-row index lookup on Professors using PRIMARY (ProfessorID=Teach.ProfessorID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=169)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(p.Rate)  (cost=633.40 rows=3152) (actual time=1.011..1.011 rows=1 loops=1)
                            -> Table scan on p  (cost=318.20 rows=3152) (actual time=0.019..0.720 rows=3152 loops=1)
```

Figure 16. EXPLAIN ANALYZE without any index(For comparison)

```
-> Sort: Professors.Rate DESC, Courses.CourseNumber  (actual time=3.177..3.185 rows=66 loops=1)
    -> Table scan on <temporary>  (cost=0.02..6.00 rows=280) (actual time=0.002..0.010 rows=66 loops=1)
        -> Temporary table with deduplication  (cost=960.44..966.42 rows=280) (actual time=3.125..3.137 rows=66 loops=1)
            -> Nested loop inner join  (cost=932.41 rows=280) (actual time=0.062..3.028 rows=66 loops=1)
                -> Nested loop inner join  (cost=731.54 rows=574) (actual time=0.055..2.743 rows=169 loops=1)
                    -> Filter: (Courses.Department = 'CS')  (cost=544.15 rows=520) (actual time=0.043..2.341 rows=169 loops=1)
                        -> Table scan on Courses  (cost=544.15 rows=5199) (actual time=0.037..1.894 rows=5366 loops=1)
                    -> Index lookup on Teach using PRIMARY (CRN=Courses.CRN)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=169)
                -> Filter: (Professors.Rate > (select #2))  (cost=0.25 rows=0) (actual time=0.002..0.002 rows=0 loops=169)
                    -> Single-row index lookup on Professors using PRIMARY (ProfessorID=Teach.ProfessorID)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=169)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(p.Rate)  (cost=633.40 rows=3152) (actual time=0.937..0.937 rows=1 loops=1)
                            -> Index scan on p using rate_idx  (cost=318.20 rows=3152) (actual time=0.033..0.653 rows=3152 loops=1)
```

Figure 17. EXPLAIN ANALYZE after index in Rate

In this example, we index the rate which we use to calculate the average of rate, the actual time is decreased. I find that in the line of temporary table with deduplication, even though the cost is more, the time is smaller. This contributes to the overall better performance.

The following two examples can also compare with Figure 16 which is without any index because each time I removed the previous index and only add one index at one time.

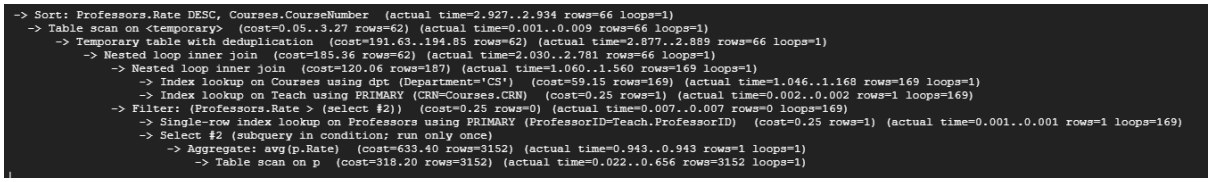

Figure 18. EXPLAIN ANALYZE after index in Department of Courses

Indexing in the Department which we use in “where” of the sql, we can gre atly decrease the time it takes, the performance is improved. This is becau se the database can directly find “CS” we need, and the cost for the filt er becomes less. This is a good way to improve the efficiency of the databa se.

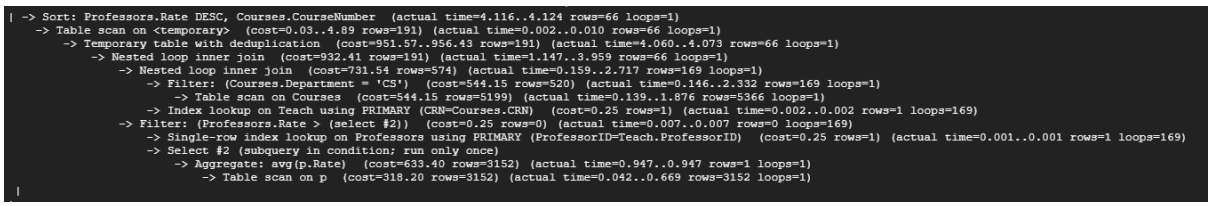

Figure 19. EXPLAIN ANALYZE after index in A of Grades

As the A in Grades is not related to this query, we can see that the perfor mance is kind of worse than the original performance, even though the diffe rence is quite minor. This is because we need extra time to set indexing wh ich we do not use in this query, so we need more time overall.