Database Implementation

Connection into GCP via Terminal

```
Welcome to Cloud Shell! Type "help" to get started.
Your Cloud Platform project in this session is set to cryptic-heaven-390921.
Use “gcloud config set project [PROJECT_ID]” to change to a different project.
maxbhartman@cloudshell:~ (cryptic-heaven-390921)$
maxbhartman@cloudshell:~ (cryptic-heaven-390921)$ gcloud sql connect cs-411-test --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 29706
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>
```

DDL Commands to create tables in Data

```sql
CREATE DATABASE IF NOT EXISTS WSPG_DB;
USE WSPG_DB;

CREATE TABLE Department (
    DepartmentName VARCHAR(255) PRIMARY KEY,
    Enrollment INT,
    CourseCount INT
    FemalePct float
);

CREATE TABLE CourseInfo (
    CourseCode VARCHAR(10) NOT NULL,
    ProfessorName VARCHAR(255) NOT NULL,
    CourseTitle VARCHAR(255),
    CourseDescription TEXT,
    Credits INT,
    DepartmentName VARCHAR(255),
    PRIMARY KEY (CourseCode, ProfessorName),
    FOREIGN KEY (DepartmentName) REFERENCES Department(DepartmentName) ON DELETE CASCADE
);

CREATE TABLE ProfessorInfo (
    ProfessorName VARCHAR(255) PRIMARY KEY,
    Number5s INT,
    Number4s INT,
    Number3s INT,
    Number2s INT,
    Number1s INT
);

CREATE TABLE AverageGPA (
    CourseCode VARCHAR(10) NOT NULL,
    ProfessorName VARCHAR(255) NOT NULL,
    NumberAPlus INT,
    NumberA INT,
    NumberAMinus INT,
    NumberBPlus INT,
    NumberB INT,
    NumberBMinus INT,
    NumberCPlus INT,
    NumberC INT,
    NumberCMinus INT,
    NumberDPlus INT,
    NumberD INT,
    NumberDMinus INT,
    NumberF INT,
    NumberW INT,
    PRIMARY KEY (CourseCode, ProfessorName),
    FOREIGN KEY (CourseCode) REFERENCES CourseInfo(CourseCode),
    FOREIGN KEY (ProfessorName) REFERENCES ProfessorInfo(ProfessorName)
);



CREATE TABLE Awards (
    ProfessorName VARCHAR(255) NOT NULL,
    Award VARCHAR(255) NOT NULL,
    Department VARCHAR(255),
    CourseCode VARCHAR(10) NOT NULL,
    Term VARCHAR(10),
    PRIMARY KEY (ProfessorName, Term),
    FOREIGN KEY (ProfessorName) REFERENCES ProfessorInfo(ProfessorName),
    FOREIGN KEY (CourseCode) REFERENCES CourseInfo(CourseCode)
);

CREATE TABLE Users (
    NetID VARCHAR(255) PRIMARY KEY,
    Major VARCHAR(255),
    GraduatingYear INT
);

CREATE TABLE MatchResult (
    NetID VARCHAR(255) NOT NULL,
    CourseCode VARCHAR(10) NOT NULL,
    Response INT,
    PRIMARY KEY (NetID, CourseCode),
    FOREIGN KEY (NetID) REFERENCES Users(NetID),
    FOREIGN KEY (CourseCode) REFERENCES CourseInfo(CourseCode)
);
```

Three tables with over 1000 rows

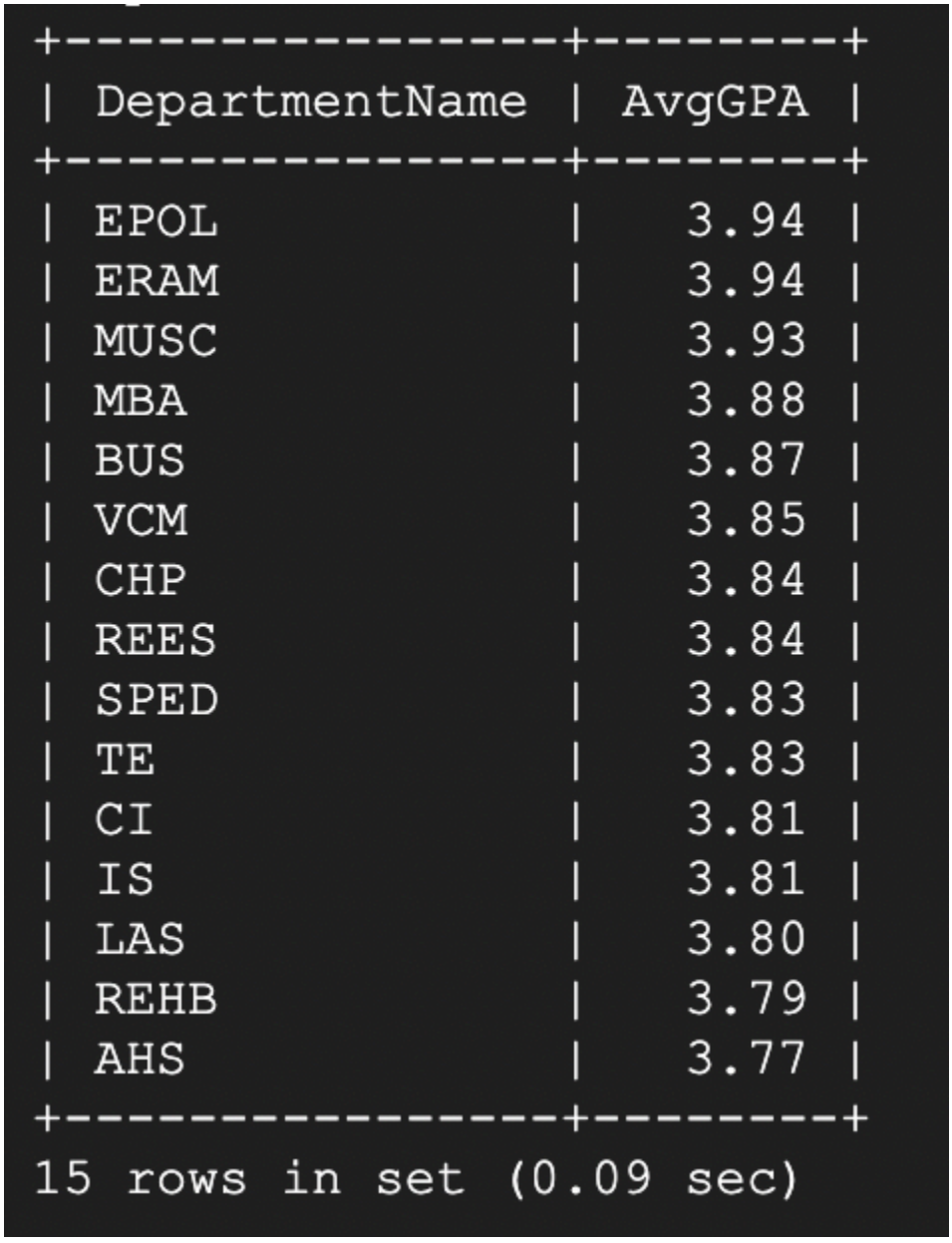
```
mysql> select count(*) from CourseInfo;
+----------+
| count(*) |
+----------+
|     5050 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> select count(*) from AverageGPA;
+----------+
| count(*) |
+----------+
|    11732 |
+----------+
1 row in set (0.01 sec)
```

```
mysql> select count(*) from ProfessorInfo;
+----------+
| count(*) |
+----------+
|     9308 |
+----------+
1 row in set (0.00 sec)
```

Queries

Average Department GPA Top 15 Results



DepartmentName	AvgGPA
EPOL	3.94
ERAM	3.94
MUSC	3.93
MBA	3.88
BUS	3.87
VCM	3.85
CHP	3.84
REES	3.84
SPED	3.83
TE	3.83
CI	3.81
IS	3.81
LAS	3.80
REHB	3.79
AHS	3.77

15 rows in set (0.09 sec)

This query computes the Average GPA for courses in each department. This query **joins** the Department, CourseInfo, and AverageGPA tables together. The query uses **Group by** to group the output AverageGPA by DepartmentName. This query will be used in our final Web App, so we can display the average gpa for the department of the class the user is prompted with.

Courses with Average GPA > 3.2 and 30% women in that Course's Department

DepartmentName	AvgGPA	FemalePct
ERAM	3.94	0.57
MUSC	3.93	0.59
MBA	3.88	0.49
BUS	3.87	0.89
VCM	3.85	0.55
REES	3.84	0.68
CHP	3.84	0.48
SPED	3.83	0.31
CI	3.81	0.67
IS	3.81	0.4
LAS	3.80	0.37
REHB	3.79	0.77
ALEC	3.77	0.57
KOR	3.76	0.63
SOCW	3.75	0.72
BDI	3.75	0.65
TRST	3.75	0.4
ENSU	3.74	0.42

This query finds departments with GPAs above 3.2 in a department that has at least 30% women enrolled. This query **joins** the Department, CourseInfo, and AverageGPA tables together. The query uses **Group by** to group the output AverageGPA by DepartmentName. This query will be used in our final Web App, so we can find departments with a high GPA (3.2+) and high diversity (30%+ women). This will be used so female users know which departments support diversity, equity, and inclusion.

Average GPA for each course

CourseCode	AvgGPA
MUSC484	3.99
IS534	3.99
MUSC480	3.98
TE510	3.98
MUSC476	3.98
FIN394	3.98
ACCY518	3.97
ENG411	3.97
ABE199	3.97
CI443	3.97
KIN449	3.97
FSHN249	3.97
IS544	3.97
IS565	3.97
ENG377	3.97

This query lists courses in order of GPA in the course. This query **joins** CourseInfo and AverageGPA tables together. The query uses **Group by** to group the output AverageGPA by CourseCode. This query is essential to our final app as this is how we will display the average gpa for a course presented to the user.

Great Professors: 3.5 Average GPA in courses taught and won awards

ProfessorName	AvgGPA
Hensley, M	3.95
Chen, M	3.95
Grift, T	3.94
Leiby, J	3.93
Craig, A	3.93
Tate, A	3.93
Parsons, C	3.92
Escobar-Sawicki, C	3.92
Napolitano, C	3.91
Rangarajan, A	3.90
Smith, N	3.90
Light Shriner, C	3.90
Mendes, C	3.89
Zarate, K	3.89
Erickson, K	3.88

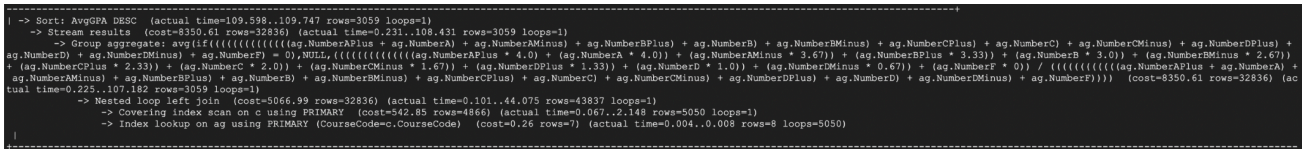
This query distinguishes the top faculty at this university by filtering professors by the average gpa in their courses and if they won any awards. This query **joins** CourseInfo, AverageGPA, and Professor tables together. The query uses **Group by** to group the output AverageGPA by ProfessorName. This query is essential to our final app as this is how we will display if the course a user is prompted for has a great professor teaching the course.

Indexing

Query 1 (Course_AvgGPA.sql)

Baseline

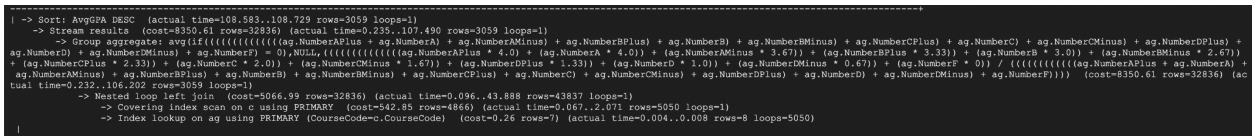
- Nested loop left join cost: 5066.99



Experiment 1

CREATE INDEX counts_idx on AverageGPA(NumberAPlus, NumberA, NumberAMinus, NumberBPlus, NumberB, NumberBMinus, NumberCPlus, NumberC, NumberCMinus, NumberDPlus, NumberD, NumberDMinus, NumberF);

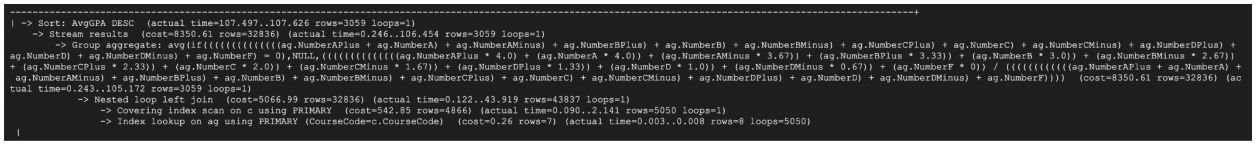
- Nested loop left join cost: 5066.99
- Compared to the baseline, this is the same.



Experiment 2

CREATE INDEX CourseCode_idx on AverageGPA(CourseCode);

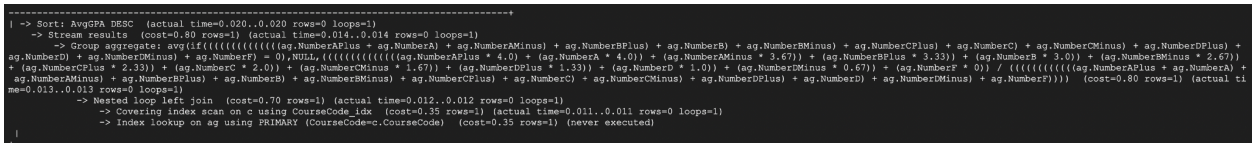
- Nested loop left join: cost = 5066.99
- Compared to the baseline, this is the same.



Experiment 3

CREATE INDEX CourseCode_idx on CourseInfo(CourseCode);

- Nested loop left join: cost = 0.70
- Compared to the baseline, the cost is lower so we have a performance gain



Final index design:

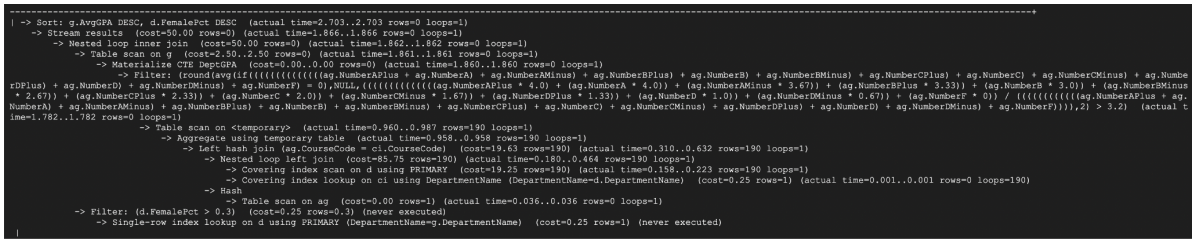
For this query, we will be using our experiment 3 (CREATE INDEX CourseCode_idx on CourseInfo(CourseCode);) as our way of indexing. The reason we are choosing this is because

our cost of 0.70 for the nested loop left join is significantly lower than our baseline and other 3 experiments, which were all 5066.99.

Query 2 (Diversity.sql):

Baseline

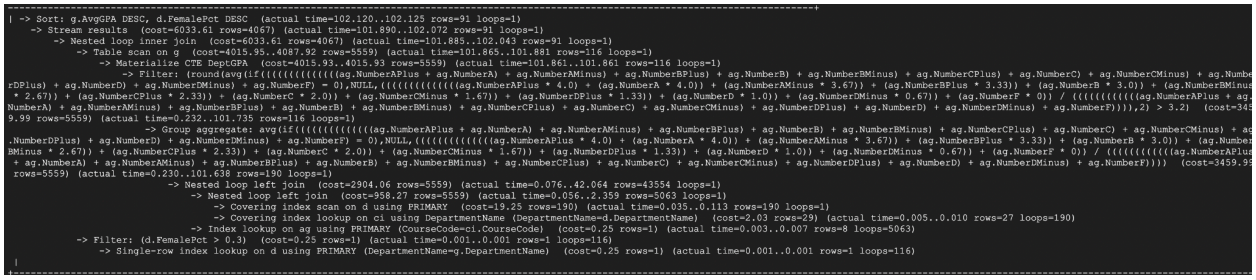
- Nested loop inner join: 50.00



Experiment 1

CREATE INDEX woman_idx on Department(FemalePct);

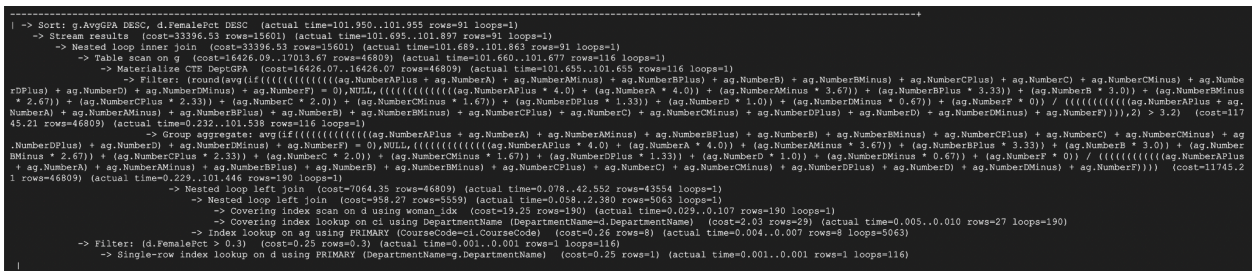
- Nested loop inner join: cost = 6033.61
- Compared to the baseline, this is a higher cost so we have a performance degradation



Experiment 2

CREATE INDEX counts_idx on AverageGPA(NumberAPlus, NumberA, NumberAMinus, NumberBPlus, NumberB, NumberBMinus, NumberCPlus, NumberC, NumberCMinus, NumberDPlus, NumberD, NumberDMinus, NumberF);

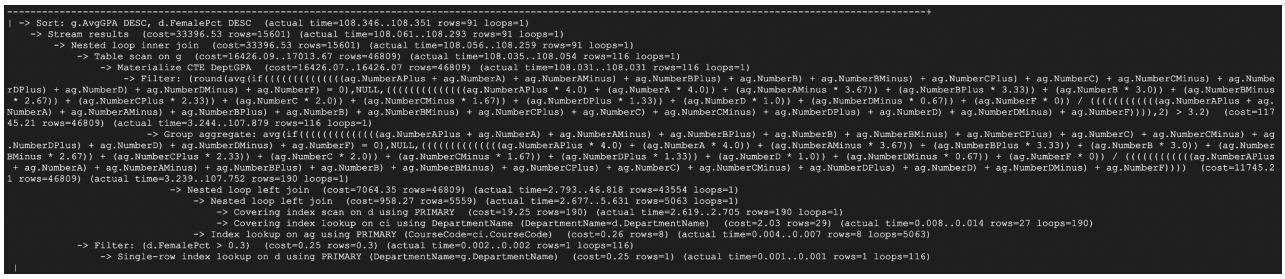
- Nested loop inner join: cost = 33396.53
- Compared to the baseline, this is a higher cost (even higher than experiment 1) so this is also a performance degradation.



Experiment 3

CREATE INDEX woman_idx on Department(DepartmentName);

- Nested loop inner join: cost = 33396.53
- Compared to the baseline, this is a higher cost (higher than experiment 1 and same as experiment 2) so this is a performance degradation



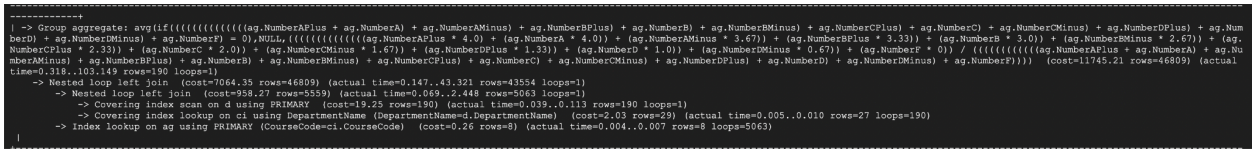
Final index design:

For this query, we will be using our baseline without indexing. The reason is because for all 3 of our experiments, our cost increased so it would make more sense for us to stick with no indexing as it led us to having the lowest cost.

Query 3 (Average GPA for Dept.)

BASELINE

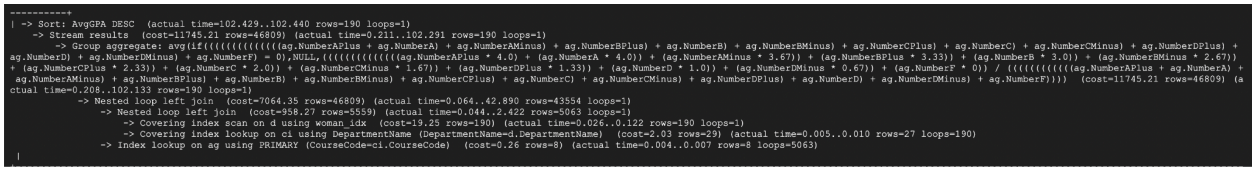
- Nested loop left join: cost = 7064.35



Experiment 1

CREATE INDEX woman_idx on Department(DepartmentName);

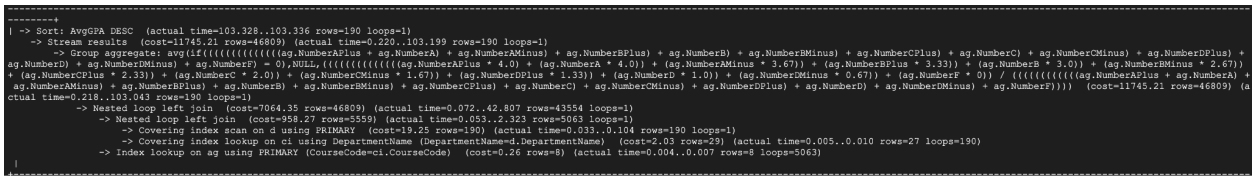
- Nested loop left join: cost = 7064.35
- Compared to the baseline, this is the same



Experiment 2

CREATE INDEX woman_idx on CourseInfo(CourseCode);

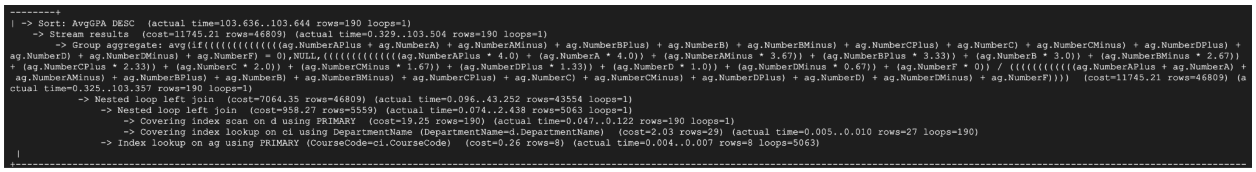
- Nested loop left join: cost = 7064.35
- Compared to the baseline, this is the same



Experiment 3

CREATE INDEX counts_idx on AverageGPA(NumberAPlus, NumberA, NumberAMinus, NumberBPlus, NumberB, NumberBMinus, NumberCPlus, NumberC, NumberCMinus, NumberDPlus, NumberD, NumberDMinus, NumberF);

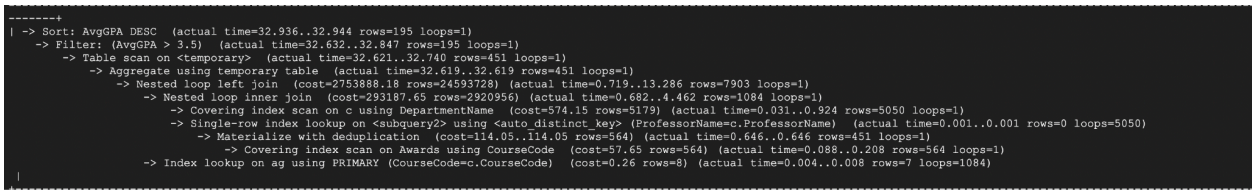
- Nested loop left join: cost = 7064.35
- Compared to the baseline, this is the same



Final index design:
For this query, we will be using our baseline without indexing. The reason is because we did not find any difference in cost when we tried adding different indices to different attributes. We think this is the case, because of the cardinality of the Department table. Changing the query index did not impact the cost, since the table is small enough where the SQL engine likely just computes a one-pass join.

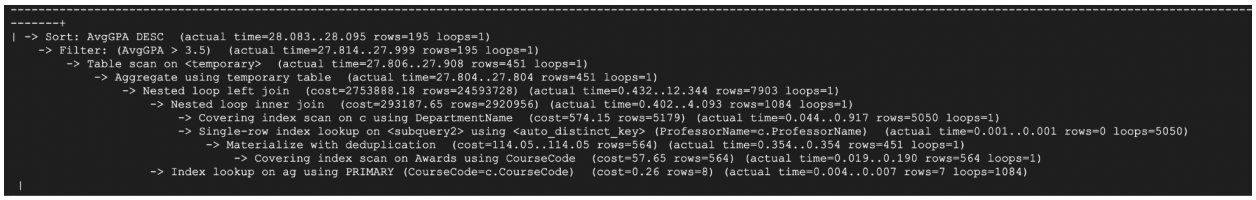
Query 4 (Professor)
Baseline

- Nested loop left join: cost = 2753888.18



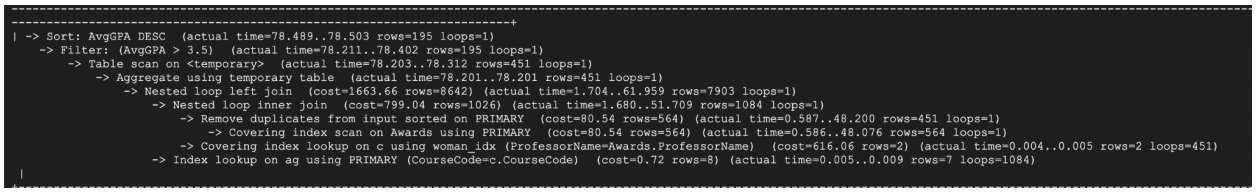
EXPERIMENT 1
CREATE INDEX counts_idx on AverageGPA(NumberAPlus, NumberA, NumberAMinus, NumberBPlus, NumberB, NumberBMinus, NumberCPlus, NumberC, NumberCMinus, NumberDPlus, NumberD, NumberDMinus, NumberF);

- Nested loop left join: cost = 2753888.18
- Compared to the baseline, this is the same



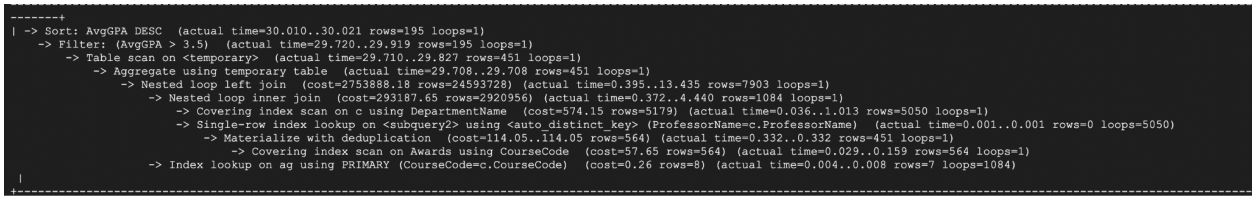
EXPERIMENT 2
CREATE INDEX counts_idx on CourseInfo(ProfessorName)

- Nested loop left join: cost = 1663.66
- Compared to the baseline, this has a significantly lower cost so we have a performance gain



EXPERIMENT 3
CREATE INDEX counts_idx on Awards(ProfessorName)

- Nested loop left join: cost = 2753888.18
- Compared to the baseline, this is the same



Final index design:

For this query, we will be using our experiment 2 (CREATE INDEX counts_idx on CourseInfo(ProfessorName)) to index. The reason is because our cost of 1663.66 for the nested loop left join is significantly lower than our cost for the baseline and other two experiments, which were all 275388.18.