CS411 Project Track1 Stage 3
Group number: 50
Group name: Hajimi

Overview:

Developing Environment: GCP
MySQLVersion: 8.0
Design Purpose: Realize the database we proposed in stage 2
We used auto-generated data.

1. Environment Setup:

MySQL 8.0 on GCP

```
cs411hajimi@cloudshell:~ (round-logic-476700-h8)$ gcloud sql connect cs411-sql-server --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 42498
Server version: 8.0.41-google (Google)

Copyright (c) 2000, 2025, 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>
```

2. Table description

Tables Overview

```
mysql> Show Tables;
+-----------------------+
| Tables_in_edu_connect |
+-----------------------+
| Bookmark              |
| Comparison            |
| Job                   |
| JobPreference         |
| Major                 |
| MajorJob              |
| MajorPreference       |
| Program               |
| University            |
| User                  |
+-----------------------+
10 rows in set (0.01 sec)
```

Table: Bookmark

```
mysql> Describe Bookmark
    -> ;
+-----------+------+------+-----+---------+-------+
| Field     | Type | Null | Key | Default | Extra |
+-----------+------+------+-----+---------+-------+
| UserID    | int  | NO   | PRI | NULL    |       |
| ProgramID | int  | NO   | PRI | NULL    |       |
+-----------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> Select count(*)
    -> from Bookmark;
+----------+
| count(*) |
+----------+
|     3000 |
+----------+
1 row in set (0.00 sec)
```

Table: Comparison

```
mysql> Describe Comparison
    -> ;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| ComparisonID | int          | NO   | PRI | NULL    | auto_increment |
| UserID       | int          | NO   | MUL | NULL    |                |
| ProgramID1   | int          | NO   | MUL | NULL    |                |
| ProgramID2   | int          | NO   | MUL | NULL    |                |
| NoteFromUser | varchar(255) | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

```
mysql> Select count(*)
    -> from Comparison;
+----------+
| count(*) |
+----------+
|     1500 |
+----------+
1 row in set (0.00 sec)
```

Table: Job

```
mysql> Describe Job;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| JobID     | int          | NO   | PRI | NULL    | auto_increment |
| JobTitle  | varchar(255) | YES  |     | NULL    |                |
| Company   | varchar(255) | YES  |     | NULL    |                |
| Location  | varchar(255) | YES  |     | NULL    |                |
| AvgSalary | int          | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

```
mysql> Select count(*)
    -> from Job;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.01 sec)
```

Table: JobPreference

```
mysql> Describe JobPreference;
+--------+------+------+-----+---------+-------+
| Field  | Type | Null | Key | Default | Extra |
+--------+------+------+-----+---------+-------+
| UserID | int  | NO   | PRI | NULL    |       |
| JobID  | int  | NO   | PRI | NULL    |       |
+--------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> Select count(*)
    -> from JobPreference;
+----------+
| count(*) |
+----------+
|     3000 |
+----------+
1 row in set (0.00 sec)
```

Table: Major

```
mysql> Describe Major;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| MajorID   | int         | NO   | PRI | NULL    | auto_increment |
| MajorName | varchar(255) | YES  |     | NULL    |                |
| Field     | varchar(255) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

```
mysql> Select count(*)
    -> from Major;
+----------+
| count(*) |
+----------+
|       30 |
+----------+
1 row in set (0.00 sec)
```

Table: MajorJob

```
mysql> Describe MajorJob;
+---------+------+------+-----+---------+-------+
| Field   | Type | Null | Key | Default | Extra |
+---------+------+------+-----+---------+-------+
| MajorID | int  | NO   | PRI | NULL    |       |
| JobID   | int  | NO   | PRI | NULL    |       |
+---------+------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

```
mysql> Select count(*)
    -> from MajorJob;
+----------+
| count(*) |
+----------+
|     2000 |
+----------+
1 row in set (0.00 sec)
```

Table: MajorPreference

```
mysql> Describe MajorPreference;
+---------+------+------+-----+---------+-------+
| Field   | Type | Null | Key | Default | Extra |
+---------+------+------+-----+---------+-------+
| UserID  | int  | NO   | PRI | NULL    |       |
| MajorID | int  | NO   | PRI | NULL    |       |
+---------+------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

```
mysql> Select count(*)
    -> from MajorPreference;
+----------+
| count(*) |
+----------+
|     3000 |
+----------+
1 row in set (0.00 sec)
```

Table: Program

```
mysql> Describe Program;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| ProgramID    | int          | NO   | PRI | NULL    | auto_increment |
| Name         | varchar(255) | YES  |     | NULL    |                |
| UniversityID | int          | NO   | MUL | NULL    |                |
| MajorID      | int          | NO   | MUL | NULL    |                |
| MedianSalary | int          | YES  |     | NULL    |                |
| DegreeType   | varchar(255) | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)
```

```
mysql> Select count(*)
    -> from Program;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)
```

Table: University

```
mysql> Describe University;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| UniversityID | int          | NO   | PRI | NULL    | auto_increment |
| Name         | varchar(255) | YES  |     | NULL    |                |
| Location     | varchar(255) | YES  |     | NULL    |                |
| Region       | varchar(255) | YES  |     | NULL    |                |
| Tuition      | int          | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

```
mysql> Select count(*)
    -> from University;
+----------+
| count(*) |
+----------+
|       80 |
+----------+
1 row in set (0.00 sec)
```

Table: User

```
mysql> Describe User;
+------------------+-------------+------+-----+---------+----------------+
| Field            | Type        | Null | Key | Default | Extra          |
+------------------+-------------+------+-----+---------+----------------+
| UserID           | int         | NO   | PRI | NULL    | auto_increment |
| Username         | varchar(255)| YES  |     | NULL    |                |
| Email            | varchar(255)| YES  |     | NULL    |                |
| PasswordHash     | varchar(255)| YES  |     | NULL    |                |
| PreferredMajor   | int         | YES  | MUL | NULL    |                |
| PreferredLocation| varchar(255)| YES  |     | NULL    |                |
| PreferredJob     | int         | YES  | MUL | NULL    |                |
+------------------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

```
mysql> Select count(*)
    -> from User;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.00 sec)
```

3. DDL command used to create table

```sql
CREATE TABLE Major (
  MajorID INT AUTO_INCREMENT PRIMARY KEY,
  MajorName VARCHAR(255),
  Field VARCHAR(255)
);

CREATE TABLE Job (
  JobID INT AUTO_INCREMENT PRIMARY KEY,
  JobTitle VARCHAR(255),
  Company VARCHAR(255),
  Location VARCHAR(255),
  AvgSalary INT
);

CREATE TABLE University (
  UniversityID INT AUTO_INCREMENT PRIMARY KEY,
  Name VARCHAR(255),
  Location VARCHAR(255),
  Region VARCHAR(255),
  Tuition INT
);
```

```sql
CREATE TABLE Program (
  ProgramID INT AUTO_INCREMENT PRIMARY KEY,
  Name VARCHAR(255),
  UniversityID INT NOT NULL,
  MajorID INT NOT NULL,
  MedianSalary INT,
  DegreeType VARCHAR(255),
  FOREIGN KEY (UniversityID)
    REFERENCES University(UniversityID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY (MajorID)
    REFERENCES Major(MajorID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);

CREATE TABLE User (
  UserID INT AUTO_INCREMENT PRIMARY KEY,
  Username VARCHAR(255),
  Email VARCHAR(255),
  PasswordHash VARCHAR(255),
  PreferredMajor INT NULL,
  PreferredLocation VARCHAR(255),
  PreferredJob INT NULL,
  FOREIGN KEY (PreferredMajor)
    REFERENCES Major(MajorID)
    ON UPDATE CASCADE
    ON DELETE SET NULL,
  FOREIGN KEY (PreferredJob)
    REFERENCES Job(JobID)
    ON UPDATE CASCADE
    ON DELETE SET NULL
);

CREATE TABLE Comparison (
  ComparisonID INT AUTO_INCREMENT PRIMARY KEY,
  UserID INT NOT NULL,
  ProgramID1 INT NOT NULL,
```

```
  ProgramID2 INT NOT NULL,
  NoteFromUser VARCHAR(255),
  FOREIGN KEY (UserID)
    REFERENCES User(UserID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY (ProgramID1)
    REFERENCES Program(ProgramID)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
  FOREIGN KEY (ProgramID2)
    REFERENCES Program(ProgramID)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);

CREATE TABLE Bookmark(
    UserID INT NOT NULL,
    ProgramID INT NOT NULL,
    PRIMARY KEY(UserID, ProgramID),
    FOREIGN KEY (UserID)
      REFERENCES User(UserID)
      ON UPDATE CASCADE
      ON DELETE CASCADE,
    FOREIGN KEY (ProgramID)
      REFERENCES Program(ProgramID)
      ON UPDATE CASCADE
      ON DELETE CASCADE
);

CREATE TABLE MajorJob(
    MajorID INT NOT NULL,
    JobID INT NOT NULL,
    PRIMARY KEY(MajorID, JobID),
    FOREIGN KEY (MajorID)
      REFERENCES Major(MajorID)
      ON UPDATE CASCADE
      ON DELETE CASCADE,
    FOREIGN KEY (JobID)
      REFERENCES Job(JobID)
```

```
      ON UPDATE CASCADE
      ON DELETE CASCADE
);

CREATE TABLE JobPreference(
    UserID INT NOT NULL,
    JobID INT NOT NULL,
    PRIMARY KEY(UserID, JobID),
    FOREIGN KEY (UserID)
      REFERENCES User(UserID)
      ON UPDATE CASCADE
      ON DELETE CASCADE,
    FOREIGN KEY (JobID)
      REFERENCES Job(JobID)
      ON UPDATE CASCADE
      ON DELETE CASCADE
);

CREATE TABLE MajorPreference(
    UserID INT NOT NULL,
    MajorID INT NOT NULL,
    PRIMARY KEY(UserID, MajorID),
    FOREIGN KEY (UserID)
      REFERENCES User(UserID)
      ON UPDATE CASCADE
      ON DELETE CASCADE,
    FOREIGN KEY (MajorID)
      REFERENCES Major(MajorID)
      ON UPDATE CASCADE
      ON DELETE CASCADE
);
```

4. Database Query Result:

Query 1: Programs that have been bookmarked the most in each field

```
SELECT
    a.ProgramID,
```

```sql
    a.ProgramName,
    a.Field,
    a.bookmark_count
FROM (
    SELECT
        p.ProgramID,
        p.Name AS ProgramName,
        m.Field,
        COUNT(*) AS bookmark_count
    FROM Bookmark b
    JOIN Program p ON b.ProgramID = p.ProgramID
    JOIN Major m ON p.MajorID = m.MajorID
    GROUP BY p.ProgramID, p.Name, m.Field
) AS a
WHERE (
    SELECT COUNT(*)
    FROM (
        SELECT
            p2.ProgramID,
            p2.Name AS ProgramName,
            m2.Field,
            COUNT(*) AS bookmark_count
        FROM Bookmark b2
        JOIN Program p2 ON b2.ProgramID = p2.ProgramID
        JOIN Major m2 ON p2.MajorID = m2.MajorID
        GROUP BY p2.ProgramID, p2.Name, m2.Field
    ) AS b
    WHERE b.Field = a.Field
      AND b.bookmark_count > a.bookmark_count
) < 15
ORDER BY a.Field, a.bookmark_count DESC;
```

ProgramID	ProgramName	Field	bookmark_count
206	M.S. in Industrial Engineering	Business	9
785	B.S. in Mechanical Engineering	Business	8
825	Ph.D. in Electrical Engineering	Business	8
984	B.S. in Materials Science and Engineering	Business	7
867	B.B.A. in Finance	Business	7
413	M.Eng. in Materials Science	Business	7
253	B.S. in Environmental Engineering	Business	7
751	M.S. in Computer Science	Business	7
474	M.S. in Electrical and Computer Engineering	Business	6
709	B.B.A. in Marketing	Business	6
140	M.S. in Statistics	Business	6
1	M.S. in Civil and Environmental Engineering	Business	6
258	Ph.D. in Electrical Engineering	Business	6
886	B.S. in Civil Engineering	Business	6
238	B.B.A. in Finance	Business	6
65	B.B.A. in Marketing	Business	6
421	B.S. in Civil Engineering	Business	6
622	B.S. in Electrical Engineering	Business	6
390	Ph.D. in Computer Science	Business	6
635	B.S. in Applied Mathematics	Business	6
257	MBA in Marketing	Business	6
6	B.A. in History	Humanities	10
909	M.S. in Computer Science	Humanities	8
551	B.A. in Sociology	Humanities	8
599	Ph.D. in Electrical Engineering	Humanities	7
410	B.S. in Applied Mathematics	Humanities	7
55	B.A. in Economics	Humanities	7

Query 2: Universities and programs that have been compared together

```
SELECT
    LEAST(u1.Name, u2.Name) AS UnivA,
    GREATEST(u1.Name, u2.Name) AS UnivB,
    p1.Name AS ProgramA,
    p2.Name AS ProgramB,
    COUNT(*) AS times_compared
FROM Comparison c
JOIN Program p1 ON c.ProgramID1 = p1.ProgramID
JOIN University u1 ON p1.UniversityID = u1.UniversityID
JOIN Program p2 ON c.ProgramID2 = p2.ProgramID
JOIN University u2 ON p2.UniversityID = u2.UniversityID
GROUP BY
    LEAST(u1.Name, u2.Name),
    GREATEST(u1.Name, u2.Name),
```

```
    p1.Name,
    p2.Name
ORDER BY
    UnivA,
    UnivB,
    times_compared DESC
LIMIT 15;
```

UnivA	UnivB	ProgramA	ProgramB	times_compared
Andrewsberg University	Brettland University	Ph.D. in Mechanical Engineering	B.B.A. in Marketing	1
Andrewsberg University	Devonshire University	B.B.A. in Marketing	B.A. in Philosophy	1
Andrewsberg University	East Alexandria University	B.S. in Mechanical Engineering	M.S. in Data Science	1
Andrewsberg University	Emilyshire University	B.S. in Environmental Engineering	B.B.A. in Marketing	1
Andrewsberg University	Gilmorefort University	B.S. in Civil Engineering	B.S. in Materials Science and Engineering	1
Andrewsberg University	Jasonview University	B.S. in Biomedical Engineering	Ph.D. in Economics	1
Andrewsberg University	Lake Jenniferview University	B.A. in Economics	B.B.A. in Marketing	1
Andrewsberg University	Lake Kayla University	B.S. in Environmental Engineering	B.B.A. in Marketing	1
Andrewsberg University	Lake Shawnshire University	B.B.A. in Marketing	MBA in Marketing	1
Andrewsberg University	Marissamouth University	B.S. in Data Science	B.B.A. in Marketing	1
Andrewsberg University	Michaelport University	M.Eng. in Materials Science	B.S. in Biomedical Engineering	1
Andrewsberg University	New Erica University	B.S. in Civil Engineering	MBA in Strategic Management	1
Andrewsberg University	New Patricialand University	B.S. in Civil Engineering	M.S. in Finance	1
Andrewsberg University	North Brianton University	M.Eng. in Materials Science	B.B.A. in Marketing	1
Andrewsberg University	Patelhaven University	M.S. in Business Analytics	B.A. in Sociology	1

Query 3: Majors that have high median salary

```
SELECT
    m.MajorName,
    m.Field,
    p.Name AS ProgramName,
    ROUND(AVG(p.MedianSalary), 2) AS Avg_Median_Salary
FROM Program p
JOIN Major m ON p.MajorID = m.MajorID
GROUP BY m.MajorID, m.MajorName, m.Field, p.Name
ORDER BY Avg_Median_Salary DESC
LIMIT 15;
```

MajorName	Field	ProgramName	Avg_Median_Salary
History	Humanities	MBA in Operations Management	149985.00
Civil Engineering	STEM	M.S. in Business Analytics	149568.00
Biomedical Engineering	STEM	MBA in Entrepreneurship	149492.00
Accounting	Business	M.S. in Computer Science	149483.00
Statistics	STEM	Ph.D. in Mechanical Engineering	149409.00
Management Information Systems	Business	B.S. in Electrical Engineering	149313.00
Industrial Engineering	STEM	B.A. in Linguistics	149291.00
Data Science	STEM	B.S. in Biomedical Engineering	149095.00
Political Science	Social Science	M.S. in Finance	148632.00
Management Information Systems	Business	B.S. in Civil Engineering	148529.00
English Literature	Humanities	M.S. in Data Science	148507.00
Industrial Engineering	STEM	B.A. in Communication Studies	148180.00
History	Humanities	B.B.A. in Supply Chain Management	148120.00
Sociology	Social Science	B.S. in Applied Mathematics	148032.00
Environmental Engineering	STEM	B.B.A. in Accounting	147464.00

Query 4: Programs that have high tuition but low median salary

```
SELECT
    p.Name AS ProgramName,
    u.Name AS University,
    m.MajorName,
    m.Field,
    p.ProgramID,
    p.DegreeType,
    u.Tuition,
    p.MedianSalary,
    ROUND(p.MedianSalary - u.Tuition, 2) AS ValueScore
FROM Program p
JOIN University u ON p.UniversityID = u.UniversityID
JOIN Major m ON p.MajorID = m.MajorID
WHERE p.MedianSalary IS NOT NULL
  AND u.Tuition > (SELECT AVG(Tuition) FROM University)
ORDER BY ValueScore ASC
LIMIT 15;
```

ProgramName	University	MajorName	Field	ProgramID	DegreeType	Tuition	MedianSalary	ValueScore
B.A. in Economics	Lake Jenniferview University	Philosophy	Humanities	513	MEng	69067	51194	-17873
B.A. in International Relations	Port Maryshire University	Communication Studies	Humanities	890	BS	69757	52397	-17360
B.B.A. in Marketing	Devonshire University	Electrical Engineering	STEM	184	MEng	71591	54621	-16970
B.B.A. in Supply Chain Management	Lake Kayla University	Materials Science and Engineering	STEM	618	PhD	71435	54625	-16810
M.S. in Computer Science	Marissamouth University	Political Science	Social Science	126	PhD	68090	52302	-15788
B.B.A. in Marketing	Millerside University	International Relations	Social Science	116	BS	71423	56756	-14667
B.S. in Civil Engineering	Andrewsberg University	Biomedical Engineering	STEM	791	MEng	72058	57608	-14450
Ph.D. in Mechanical Engineering	New Rogerburgh University	Computer Science	STEM	475	BEng	72588	58400	-14188
M.S. in Data Science	Devonshire University	Accounting	Business	334	MS	71591	57546	-14045
B.B.A. in Finance	East Mark University	Chemical Engineering	STEM	915	BS	72971	59967	-13004
M.S. in Computer Science	East Alexandria University	Linguistics	Humanities	909	MBA	66397	54192	-12205
B.B.A. in Finance	Devonshire University	Entrepreneurship	Business	238	BEng	71591	59941	-11650
M.S. in Statistics	Marissamouth University	Biomedical Engineering	STEM	260	BEng	68090	56464	-11626
B.B.A. in Marketing	Andrewsberg University	Supply Chain Management	Business	872	BS	72058	60831	-11227
B.S. in Environmental Engineering	Port Michelleburgh University	Environmental Engineering	STEM	216	MEng	63563	52409	-11154

Analyze Result:

Query 1 without any index:

```
| -> Sort: a.Field, a.bookmark_count DESC  (cost=2.6..2.6 rows=0) (actual time=168..168 rows=71 loops=1)
    -> Filter: ((select #3) < 15)  (cost=2.5..2.5 rows=0) (actual time=13.3..168 rows=71 loops=1)
        -> Table scan on a  (cost=2.5..2.5 rows=0) (actual time=6.12..6.24 rows=941 loops=1)
            -> Materialize  (cost=0..0 rows=0) (actual time=6.12..6.12 rows=941 loops=1)
                -> Table scan on <temporary>  (actual time=5.79..5.91 rows=941 loops=1)
                    -> Aggregate using temporary table  (actual time=5.79..5.79 rows=941 loops=1)
                        -> Nested loop inner join  (cost=808 rows=3188) (actual time=0.0351..3.4 rows=3000 loops=1)
                            -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0283..1.18 rows=1000 loops=1)
                                -> Table scan on m  (cost=3.25 rows=30) (actual time=0.01..0.0213 rows=30 loops=1)
                                -> Index lookup on p using idx_program_major (MajorID=m.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00625..0.0371 rows=33
.3 loops=30)
                            -> Covering index lookup on b using idx_bookmark_program_id (ProgramID=p.ProgramID)  (cost=0.251 rows=3.19) (actual time=0.00142
..0.00191 rows=3 loops=1000)
        -> Select #3 (subquery in condition; dependent)
            -> Aggregate: count(0)  (cost=63.6..63.6 rows=1) (actual time=0.171..0.171 rows=1 loops=941)
                -> Filter: ((b.Field = a.Field) and (b.bookmark_count > a.bookmark_count))  (cost=0.499..53 rows=106) (actual time=0.0288..0.166 rows=113 lo
ops=941)
                    -> Table scan on b  (cost=2.5..2.5 rows=0) (actual time=0.00638..0.0949 rows=941 loops=941)
                        -> Materialize  (cost=0..0 rows=0) (actual time=5.79..5.79 rows=941 loops=1)
                            -> Table scan on <temporary>  (actual time=5.49..5.59 rows=941 loops=1)
                                -> Aggregate using temporary table  (actual time=5.49..5.49 rows=941 loops=1)
                                    -> Nested loop inner join  (cost=808 rows=3188) (actual time=0.0225..3.15 rows=3000 loops=1)
                                        -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0182..0.968 rows=1000 loops=1)
                                            -> Table scan on m2  (cost=3.25 rows=30) (actual time=0.00743..0.0162 rows=30 loops=1)
                                            -> Index lookup on p2 using idx_program_major (MajorID=m2.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00469..
0.0301 rows=33.3 loops=30)
                                        -> Covering index lookup on b2 using idx_bookmark_program_id (ProgramID=p2.ProgramID)  (cost=0.251 rows=3.19) (actua
l time=0.00139..0.00188 rows=3 loops=1000)
 |
```

With index on Program.Name:

```
-> Sort: a.Field, a.bookmark_count DESC  (cost=2.6..2.6 rows=0) (actual time=175..175 rows=71 loops=1)
    -> Filter: ((select #3) < 15)  (cost=2.5..2.5 rows=0) (actual time=13.4..175 rows=71 loops=1)
        -> Table scan on a  (cost=2.5..2.5 rows=0) (actual time=6..6.12 rows=941 loops=1)
            -> Materialize  (cost=0..0 rows=0) (actual time=6..6 rows=941 loops=1)
                -> Table scan on <temporary>  (actual time=5.69..5.8 rows=941 loops=1)
                    -> Aggregate using temporary table  (actual time=5.69..5.69 rows=941 loops=1)
                        -> Nested loop inner join  (cost=808 rows=3188) (actual time=0.0326..3.35 rows=3000 loops=1)
                            -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0266..1.13 rows=1000 loops=1)
                                -> Table scan on m  (cost=3.25 rows=30) (actual time=0.00956..0.0198 rows=30 loops=1)
                                -> Index lookup on p using idx_program_major (MajorID=m.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00551..0.0355 rows=33
.3 loops=30)
                            -> Covering index lookup on b using idx_bookmark_program_id (ProgramID=p.ProgramID)  (cost=0.251 rows=3.19) (actual time=0.00142
..0.0019 rows=3 loops=1000)
        -> Select #3 (subquery in condition; dependent)
            -> Aggregate: count(0)  (cost=63.6..63.6 rows=1) (actual time=0.179..0.179 rows=1 loops=941)
                -> Filter: ((b.Field = a.Field) and (b.bookmark_count > a.bookmark_count))  (cost=0.499..53 rows=106) (actual time=0.0303..0.175 rows=113 lo
ops=941)
                    -> Table scan on b  (cost=2.5..2.5 rows=0) (actual time=0.00651..0.0973 rows=941 loops=941)
                        -> Materialize  (cost=0..0 rows=0) (actual time=5.91..5.91 rows=941 loops=1)
                            -> Table scan on <temporary>  (actual time=5.6..5.72 rows=941 loops=1)
                                -> Aggregate using temporary table  (actual time=5.6..5.6 rows=941 loops=1)
                                    -> Nested loop inner join  (cost=808 rows=3188) (actual time=0.0162..3.23 rows=3000 loops=1)
                                        -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0128..0.997 rows=1000 loops=1)
                                            -> Table scan on m2  (cost=3.25 rows=30) (actual time=0.00448..0.0138 rows=30 loops=1)
                                            -> Index lookup on p2 using idx_program_major (MajorID=m2.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00471..
0.0311 rows=33.3 loops=30)
                                        -> Covering index lookup on b2 using idx_bookmark_program_id (ProgramID=p2.ProgramID)  (cost=0.251 rows=3.19) (actua
l time=0.00141..0.00191 rows=3 loops=1000)
 |
```

With index on Major.Field and Program.Name:

```
-> Sort: a.Field, a.bookmark_count DESC  (cost=2.6..2.6 rows=0) (actual time=174..174 rows=71 loops=1)
    -> Filter: ((select #3) < 15)  (cost=2.5..2.5 rows=0) (actual time=11.8..174 rows=71 loops=1)
        -> Table scan on a  (cost=2.5..2.5 rows=0) (actual time=5.95..6.09 rows=941 loops=1)
            -> Materialize  (cost=0..0 rows=0) (actual time=5.95..5.95 rows=941 loops=1)
                -> Table scan on <temporary>  (actual time=5.65..5.75 rows=941 loops=1)
                    -> Aggregate using temporary table  (actual time=5.65..5.65 rows=941 loops=1)
                        -> Nested loop inner join  (cost=808 rows=3188) (actual time=0.0501..3.29 rows=3000 loops=1)
                            -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0403..1.16 rows=1000 loops=1)
                                -> Covering index scan on m using major_field  (cost=3.25 rows=30) (actual time=0.0177..0.0267 rows=30 loops=1)
                                -> Index lookup on p using idx_program_major (MajorID=m.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00592..0.0361 rows=33
.3 loops=30)
                            -> Covering index lookup on b using idx_bookmark_program_id (ProgramID=p.ProgramID)  (cost=0.251 rows=3.19) (actual time=0.00141
..0.00189 rows=3 loops=1000)
        -> Select #3 (subquery in condition; dependent)
            -> Aggregate: count(0)  (cost=63.6..63.6 rows=1) (actual time=0.178..0.178 rows=1 loops=941)
                -> Filter: ((b.Field = a.Field) and (b.bookmark_count > a.bookmark_count))  (cost=0.499..53 rows=106) (actual time=0.0632..0.174 rows=113 lo
ops=941)
                    -> Table scan on b  (cost=2.5..2.5 rows=0) (actual time=0.0063..0.0931 rows=941 loops=941)
                        -> Materialize  (cost=0..0 rows=0) (actual time=5.69..5.69 rows=941 loops=1)
                            -> Table scan on <temporary>  (actual time=5.41..5.5 rows=941 loops=1)
                                -> Aggregate using temporary table  (actual time=5.41..5.41 rows=941 loops=1)
                                    -> Nested loop inner join  (cost=808 rows=3188) (actual time=0.0176..3.1 rows=3000 loops=1)
                                        -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0137..0.971 rows=1000 loops=1)
                                            -> Covering index scan on m2 using major_field  (cost=3.25 rows=30) (actual time=0.0052..0.0129 rows=30 loops=1)
                                            -> Index lookup on p2 using idx_program_major (MajorID=m2.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00449..
0.0304 rows=33.3 loops=30)
                                        -> Covering index lookup on b2 using idx_bookmark_program_id (ProgramID=p2.ProgramID)  (cost=0.251 rows=3.19) (actua
l time=0.00138..0.00187 rows=3 loops=1000)
 |
```

With index on Major.Field:

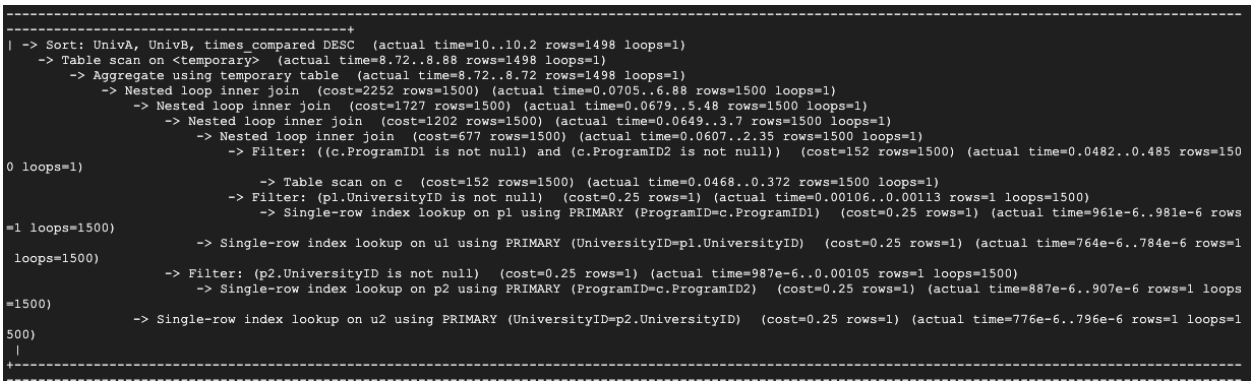
```
-> Sort: a.Field, a.bookmark_count DESC  (cost=2.6..2.6 rows=0) (actual time=160..160 rows=71 loops=1)
    -> Filter: ((select #3) < 15)  (cost=2.5..2.5 rows=0) (actual time=11.9..160 rows=71 loops=1)
        -> Table scan on a  (cost=2.5..2.5 rows=0) (actual time=5.95..6.08 rows=941 loops=1)
            -> Materialize  (cost=0..0 rows=0) (actual time=5.94..5.94 rows=941 loops=1)
                -> Table scan on <temporary>  (actual time=5.65..5.75 rows=941 loops=1)
                    -> Aggregate using temporary table  (actual time=5.64..5.64 rows=941 loops=1)
                        -> Nested loop inner join  (cost=808 rows=3188) (actual time=0.0331..3.32 rows=3000 loops=1)
                            -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.026..1.17 rows=1000 loops=1)
                                -> Covering index scan on m using major_field  (cost=3.25 rows=30) (actual time=0.00939..0.0191 rows=30 loops=1)
                                -> Index lookup on p using idx_program_major (MajorID=m.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00591..0.0367 rows=33
.3 loops=30)
                            -> Covering index lookup on b using idx_bookmark_program_id (ProgramID=p.ProgramID)  (cost=0.251 rows=3.19) (actual time=0.00141
..0.00189 rows=3 loops=1000)
        -> Select #3 (subquery in condition; dependent)
            -> Aggregate: count(0)  (cost=63.6..63.6 rows=1) (actual time=0.163..0.163 rows=1 loops=941)
                -> Filter: ((b.Field = a.Field) and (b.bookmark_count > a.bookmark_count))  (cost=0.499..53 rows=106) (actual time=0.0557..0.159 rows=113 lo
ops=941)
                    -> Table scan on b  (cost=2.5..2.5 rows=0) (actual time=0.00637..0.0882 rows=941 loops=941)
                        -> Materialize  (cost=0..0 rows=0) (actual time=5.75..5.75 rows=941 loops=1)
                            -> Table scan on <temporary>  (actual time=5.46..5.55 rows=941 loops=1)
                                -> Aggregate using temporary table  (actual time=5.46..5.46 rows=941 loops=1)
                                    -> Nested loop inner join  (cost=808 rows=3188) (actual time=0.0173..3.12 rows=3000 loops=1)
                                        -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0135..0.978 rows=1000 loops=1)
                                            -> Covering index scan on m2 using major_field  (cost=3.25 rows=30) (actual time=0.00546..0.0136 rows=30 loops=1
)
                                            -> Index lookup on p2 using idx_program_major (MajorID=m2.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00443..
0.0306 rows=33.3 loops=30)
                                        -> Covering index lookup on b2 using idx_bookmark_program_id (ProgramID=p2.ProgramID)  (cost=0.251 rows=3.19) (actua
l time=0.00139..0.00189 rows=3 loops=1000)
 |
```

Based on the result shown above, we will keep **only the Major(Field) index**. This is because the query's ranking logic is per-Field. The Field index makes the repeated "same Field" checks cheaper during the correlated subquery and reduces inner join work, producing a consistent runtime reduction in EXPLAIN ANALYZE. While for Program.Name index, it is not used for filtering or joining. And it does not affect the dominant costs.

Query 2 without index:

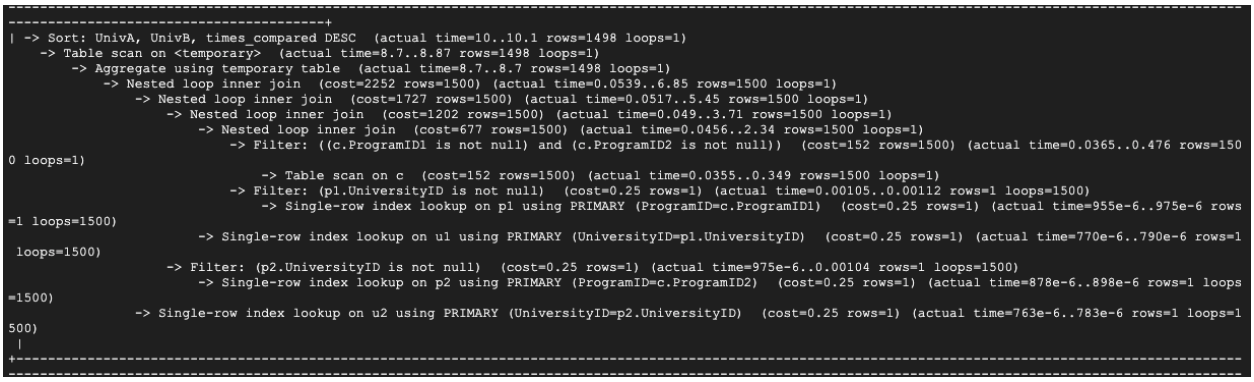
```
-> Sort: UnivA, UnivB, times_compared DESC  (actual time=10.1..10.2 rows=1498 loops=1)
    -> Table scan on <temporary>  (actual time=8.76..8.92 rows=1498 loops=1)
        -> Aggregate using temporary table  (actual time=8.75..8.75 rows=1498 loops=1)
            -> Nested loop inner join  (cost=2252 rows=1500) (actual time=0.0827..6.93 rows=1500 loops=1)
                -> Nested loop inner join  (cost=1727 rows=1500) (actual time=0.0689..5.49 rows=1500 loops=1)
                    -> Nested loop inner join  (cost=1202 rows=1500) (actual time=0.0651..3.71 rows=1500 loops=1)
                        -> Nested loop inner join  (cost=677 rows=1500) (actual time=0.0613..2.35 rows=1500 loops=1)
                            -> Filter: ((c.ProgramID1 is not null) and (c.ProgramID2 is not null))  (cost=152 rows=1500) (actual time=0.0494..0.479 rows=150
0 loops=1)
                                -> Table scan on c  (cost=152 rows=1500) (actual time=0.0478..0.365 rows=1500 loops=1)
                            -> Filter: (p1.UniversityID is not null)  (cost=0.25 rows=1) (actual time=0.00106..0.00112 rows=1 loops=1500)
                                -> Single-row index lookup on p1 using PRIMARY (ProgramID=c.ProgramID1)  (cost=0.25 rows=1) (actual time=965e-6..985e-6 rows
=1 loops=1500)
                        -> Single-row index lookup on u1 using PRIMARY (UniversityID=p1.UniversityID)  (cost=0.25 rows=1) (actual time=775e-6..795e-6 rows=1
 loops=1500)
                    -> Filter: (p2.UniversityID is not null)  (cost=0.25 rows=1) (actual time=995e-6..0.00106 rows=1 loops=1500)
                        -> Single-row index lookup on p2 using PRIMARY (ProgramID=c.ProgramID2)  (cost=0.25 rows=1) (actual time=897e-6..916e-6 rows=1 loops
=1500)
                -> Single-row index lookup on u2 using PRIMARY (UniversityID=p2.UniversityID)  (cost=0.25 rows=1) (actual time=777e-6..797e-6 rows=1 loops=1
500)
 |
```

With index on University.Name:



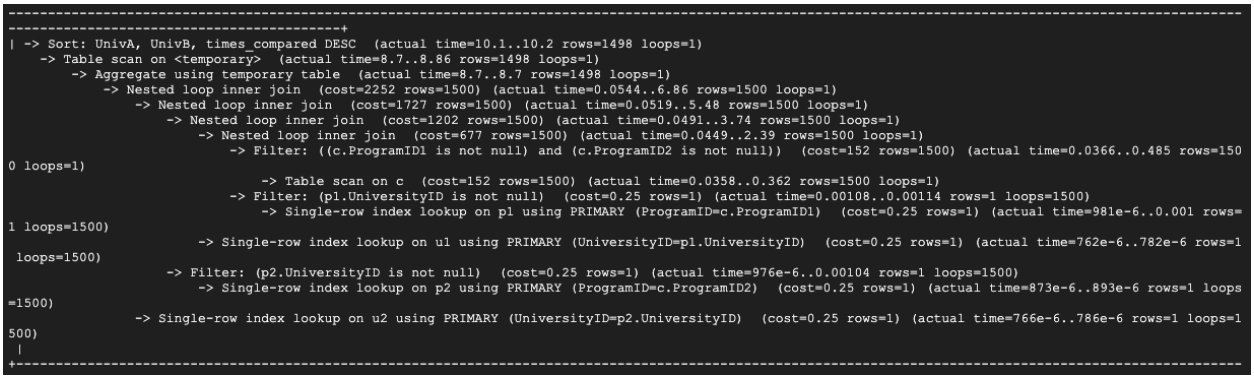
```
-> Sort: UnivA, UnivB, times_compared DESC  (actual time=10..10.2 rows=1498 loops=1)
    -> Table scan on <temporary>  (actual time=8.72..8.88 rows=1498 loops=1)
        -> Aggregate using temporary table  (actual time=8.72..8.72 rows=1498 loops=1)
            -> Nested loop inner join  (cost=2252 rows=1500) (actual time=0.0705..6.88 rows=1500 loops=1)
                -> Nested loop inner join  (cost=1727 rows=1500) (actual time=0.0679..5.48 rows=1500 loops=1)
                    -> Nested loop inner join  (cost=1202 rows=1500) (actual time=0.0649..3.7 rows=1500 loops=1)
                        -> Nested loop inner join  (cost=677 rows=1500) (actual time=0.0607..2.35 rows=1500 loops=1)
                            -> Filter: ((c.ProgramID1 is not null) and (c.ProgramID2 is not null))  (cost=152 rows=1500) (actual time=0.0482..0.485 rows=150
0 loops=1)
                                -> Table scan on c  (cost=152 rows=1500) (actual time=0.0468..0.372 rows=1500 loops=1)
                            -> Filter: (p1.UniversityID is not null)  (cost=0.25 rows=1) (actual time=0.00106..0.00113 rows=1 loops=1500)
                                -> Single-row index lookup on p1 using PRIMARY (ProgramID=c.ProgramID1)  (cost=0.25 rows=1) (actual time=961e-6..981e-6 rows
=1 loops=1500)
                        -> Single-row index lookup on u1 using PRIMARY (UniversityID=p1.UniversityID)  (cost=0.25 rows=1) (actual time=764e-6..784e-6 rows=1
 loops=1500)
                    -> Filter: (p2.UniversityID is not null)  (cost=0.25 rows=1) (actual time=987e-6..0.00105 rows=1 loops=1500)
                        -> Single-row index lookup on p2 using PRIMARY (ProgramID=c.ProgramID2)  (cost=0.25 rows=1) (actual time=887e-6..907e-6 rows=1 loops
=1500)
                -> Single-row index lookup on u2 using PRIMARY (UniversityID=p2.UniversityID)  (cost=0.25 rows=1) (actual time=776e-6..796e-6 rows=1 loops=1
500)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
```

With index on Program.Name:



```
-> Sort: UnivA, UnivB, times_compared DESC  (actual time=10..10.1 rows=1498 loops=1)
    -> Table scan on <temporary>  (actual time=8.7..8.87 rows=1498 loops=1)
        -> Aggregate using temporary table  (actual time=8.7..8.7 rows=1498 loops=1)
            -> Nested loop inner join  (cost=2252 rows=1500) (actual time=0.0539..6.85 rows=1500 loops=1)
                -> Nested loop inner join  (cost=1727 rows=1500) (actual time=0.0517..5.45 rows=1500 loops=1)
                    -> Nested loop inner join  (cost=1202 rows=1500) (actual time=0.049..3.71 rows=1500 loops=1)
                        -> Nested loop inner join  (cost=677 rows=1500) (actual time=0.0456..2.34 rows=1500 loops=1)
                            -> Filter: ((c.ProgramID1 is not null) and (c.ProgramID2 is not null))  (cost=152 rows=1500) (actual time=0.0365..0.476 rows=150
0 loops=1)
                                -> Table scan on c  (cost=152 rows=1500) (actual time=0.0355..0.349 rows=1500 loops=1)
                            -> Filter: (p1.UniversityID is not null)  (cost=0.25 rows=1) (actual time=0.00105..0.00112 rows=1 loops=1500)
                                -> Single-row index lookup on p1 using PRIMARY (ProgramID=c.ProgramID1)  (cost=0.25 rows=1) (actual time=955e-6..975e-6 rows
=1 loops=1500)
                        -> Single-row index lookup on u1 using PRIMARY (UniversityID=p1.UniversityID)  (cost=0.25 rows=1) (actual time=770e-6..790e-6 rows=1
 loops=1500)
                    -> Filter: (p2.UniversityID is not null)  (cost=0.25 rows=1) (actual time=975e-6..0.00104 rows=1 loops=1500)
                        -> Single-row index lookup on p2 using PRIMARY (ProgramID=c.ProgramID2)  (cost=0.25 rows=1) (actual time=878e-6..898e-6 rows=1 loops
=1500)
                -> Single-row index lookup on u2 using PRIMARY (UniversityID=p2.UniversityID)  (cost=0.25 rows=1) (actual time=763e-6..783e-6 rows=1 loops=1
500)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
```

With index on Program.Name and University.Name:



```
-> Sort: UnivA, UnivB, times_compared DESC  (actual time=10.1..10.2 rows=1498 loops=1)
    -> Table scan on <temporary>  (actual time=8.7..8.86 rows=1498 loops=1)
        -> Aggregate using temporary table  (actual time=8.7..8.7 rows=1498 loops=1)
            -> Nested loop inner join  (cost=2252 rows=1500) (actual time=0.0544..6.86 rows=1500 loops=1)
                -> Nested loop inner join  (cost=1727 rows=1500) (actual time=0.0519..5.48 rows=1500 loops=1)
                    -> Nested loop inner join  (cost=1202 rows=1500) (actual time=0.0491..3.74 rows=1500 loops=1)
                        -> Nested loop inner join  (cost=677 rows=1500) (actual time=0.0449..2.39 rows=1500 loops=1)
                            -> Filter: ((c.ProgramID1 is not null) and (c.ProgramID2 is not null))  (cost=152 rows=1500) (actual time=0.0366..0.485 rows=150
0 loops=1)
                                -> Table scan on c  (cost=152 rows=1500) (actual time=0.0358..0.362 rows=1500 loops=1)
                            -> Filter: (p1.UniversityID is not null)  (cost=0.25 rows=1) (actual time=0.00108..0.00114 rows=1 loops=1500)
                                -> Single-row index lookup on p1 using PRIMARY (ProgramID=c.ProgramID1)  (cost=0.25 rows=1) (actual time=981e-6..0.001 rows=
1 loops=1500)
                        -> Single-row index lookup on u1 using PRIMARY (UniversityID=p1.UniversityID)  (cost=0.25 rows=1) (actual time=762e-6..782e-6 rows=1
 loops=1500)
                    -> Filter: (p2.UniversityID is not null)  (cost=0.25 rows=1) (actual time=976e-6..0.00104 rows=1 loops=1500)
                        -> Single-row index lookup on p2 using PRIMARY (ProgramID=c.ProgramID2)  (cost=0.25 rows=1) (actual time=873e-6..893e-6 rows=1 loops
=1500)
                -> Single-row index lookup on u2 using PRIMARY (UniversityID=p2.UniversityID)  (cost=0.25 rows=1) (actual time=766e-6..786e-6 rows=1 loops=1
500)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
```

Based on the result shown above, we will **not keep any secondary indexes for this query**. EXPLAIN ANALYZE with University(Name),Program(Name), and both together shows **no change** in the dominant operators-the Aggregate using temporary table and the final Sort(times_compared DESC) have essentially identical times and rows examined compared to baseline. This might be due to the fact that Name columns are

neither join nor filter keys here, these indexes add maintenance cost without reducing work.

Query 3 without index:

```
-> Sort: Avg_Median_Salary DESC  (actual time=2.67..2.72 rows=690 loops=1)
    -> Table scan on <temporary>  (actual time=2.3..2.38 rows=690 loops=1)
        -> Aggregate using temporary table  (actual time=2.3..2.3 rows=690 loops=1)
            -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0314..1.17 rows=1000 loops=1)
                -> Table scan on m  (cost=3.25 rows=30) (actual time=0.0105..0.0215 rows=30 loops=1)
                -> Index lookup on p using idx_program_major (MajorID=m.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00603..0.0364 rows=33.3 loops=30)
```

With index on Program.Mediansalary:

```
-> Sort: Avg_Median_Salary DESC  (actual time=2.62..2.67 rows=690 loops=1)
    -> Table scan on <temporary>  (actual time=2.29..2.36 rows=690 loops=1)
        -> Aggregate using temporary table  (actual time=2.29..2.29 rows=690 loops=1)
            -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0429..1.15 rows=1000 loops=1)
                -> Table scan on m  (cost=3.25 rows=30) (actual time=0.0187..0.0297 rows=30 loops=1)
                -> Index lookup on p using idx_program_major (MajorID=m.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00624..0.0352 rows=33.3 loops=30)
```

With index on Major.MajorName:

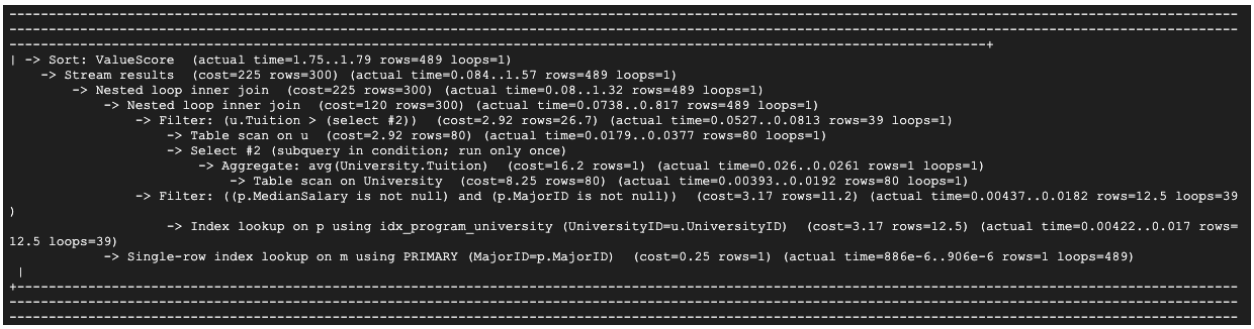
```
-> Sort: Avg_Median_Salary DESC  (actual time=2.63..2.68 rows=690 loops=1)
    -> Table scan on <temporary>  (actual time=2.28..2.37 rows=690 loops=1)
        -> Aggregate using temporary table  (actual time=2.28..2.28 rows=690 loops=1)
            -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0427..1.14 rows=1000 loops=1)
                -> Table scan on m  (cost=3.25 rows=30) (actual time=0.0183..0.0293 rows=30 loops=1)
                -> Index lookup on p using idx_program_major (MajorID=m.MajorID)  (cost=4.61 rows=33.3) (actual time=0.0059..0.0348 rows=33.3 loops=30)
```

With index on Major.MajorName and Program.MedianSalary:

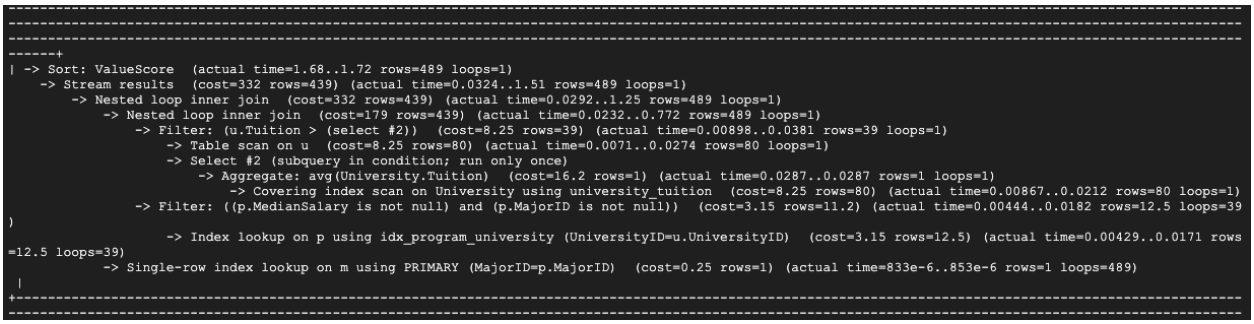
```
-> Sort: Avg_Median_Salary DESC  (actual time=2.61..2.65 rows=690 loops=1)
    -> Table scan on <temporary>  (actual time=2.27..2.34 rows=690 loops=1)
        -> Aggregate using temporary table  (actual time=2.27..2.27 rows=690 loops=1)
            -> Nested loop inner join  (cost=238 rows=1000) (actual time=0.0233..1.14 rows=1000 loops=1)
                -> Table scan on m  (cost=3.25 rows=30) (actual time=0.008..0.0192 rows=30 loops=1)
                -> Index lookup on p using idx_program_major (MajorID=m.MajorID)  (cost=4.61 rows=33.3) (actual time=0.00606..0.0354 rows=33.3 loops=30)
```

Based on the result shown above, we will keep the index **Program(MedianSalary) and Major(MajorName)**.Since The MedianSalary index will narrow I/O for the AVG aggregation, lowering the “Aggregate using temporary table” time, while the MajorName speeds the join of grouped keys. In combination they deliver the best improvement.

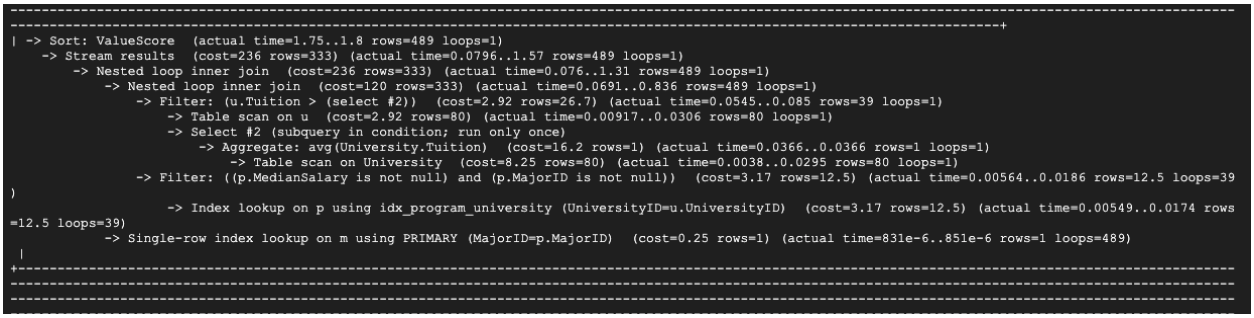
Query 4 without index:



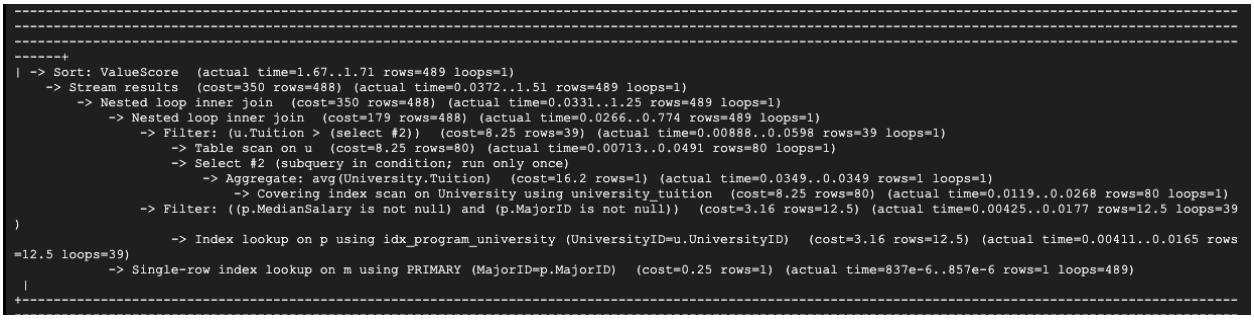
With index on University.tuition:



With index on Program.MedianSalary:



With index on University.tuition and Program.MedianSalary:



Based on the result shown above, we will keep **the index University(Tuition) and drop Program(MedianSalary).** Since EXPLAIN ANALYZE shows the biggest

improvement when indexing University(Tuition) which is the plan switches to a covering index scan for the AVG(Tuition) subquery and the Tuition> AVG range check, cutting the total time. While the Program(MedianSalary) index alone doesn’t reduce the dominant operators and adding it on top of University(Tuition) only yields a negligible improvement.