**CS411 Project Track1 Stage 3**

**Group number: 50**

**Group name: Hajimi**

**Overview:**

Developing Environment: GCP

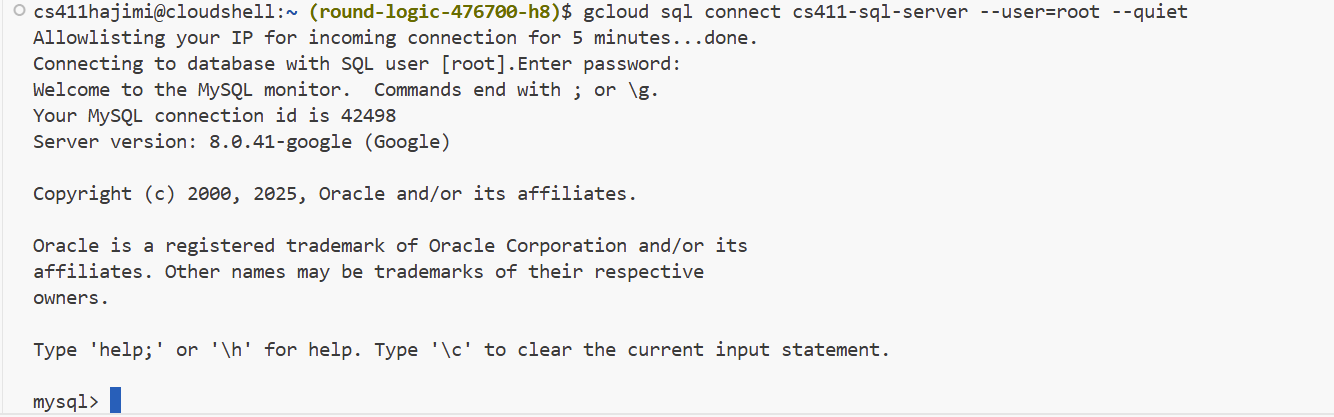
MySQLVersion: 8.0

Design Purpose: Realize the database we proposed in stage 2

We used auto-generated data.

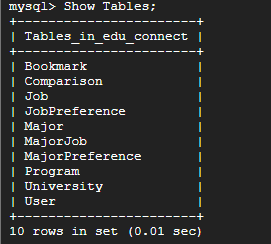
## 1. Environment Setup:

MySQL 8.0 on GCP

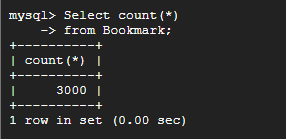
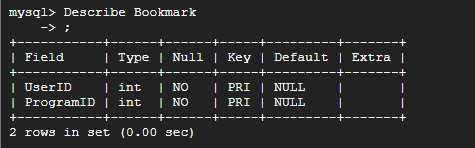


## 2. Table description

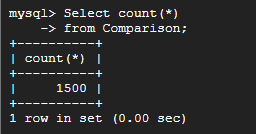
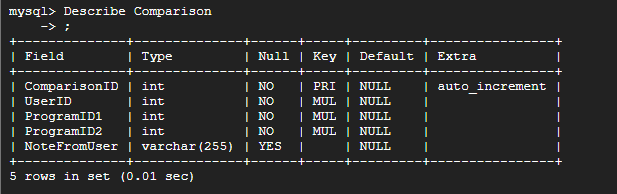
#### Tables Overview



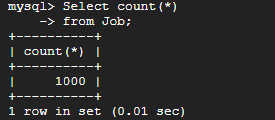
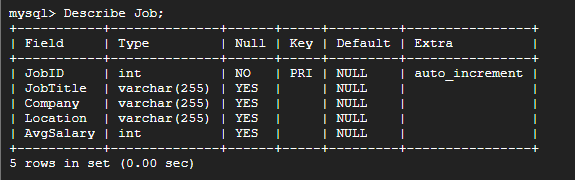
#### Table: Bookmark



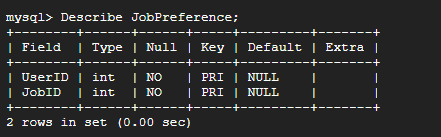
#### Table: Comparison

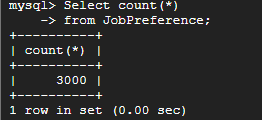


#### Table: Job

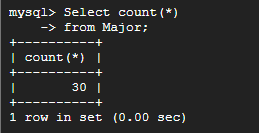
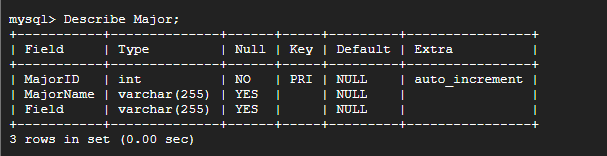


#### Table: JobPreference

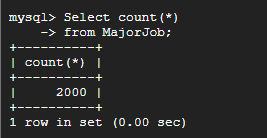
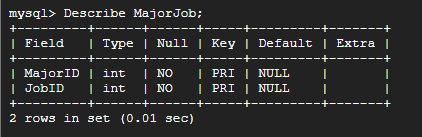
\



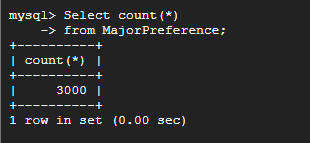
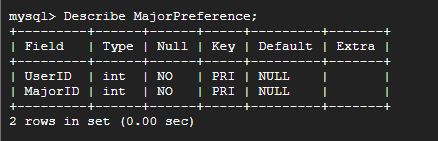
#### Table: Major



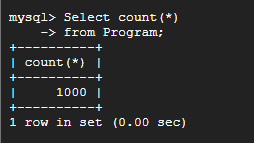
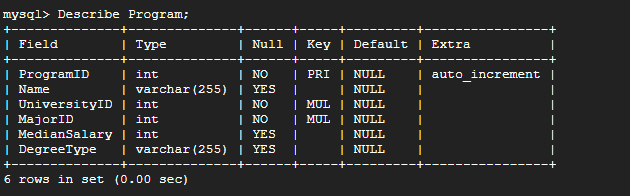
#### Table: MajorJob



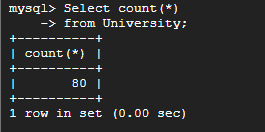
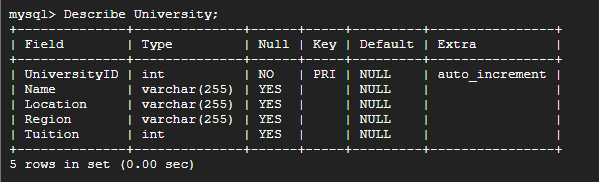
#### Table: MajorPreference



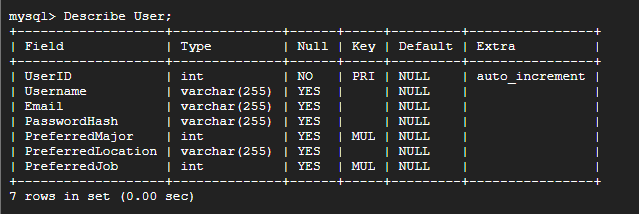
#### Table: Program

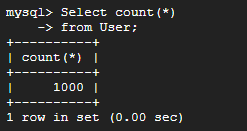


#### Table: University



#### Table: User





## 3. DDL command used to create table

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

);

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,

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

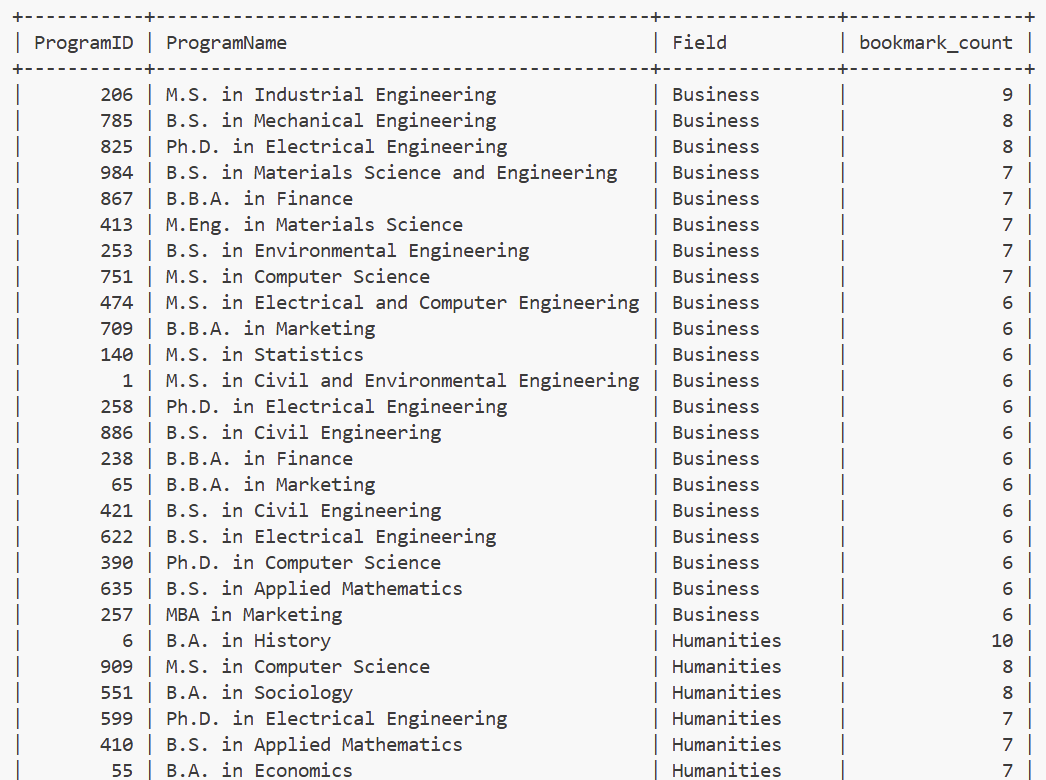
) AS b

WHERE b.Field = a.Field

AND b.bookmark\_count > a.bookmark\_count

) < 15

ORDER BY a.Field, a.bookmark\_count DESC;



**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

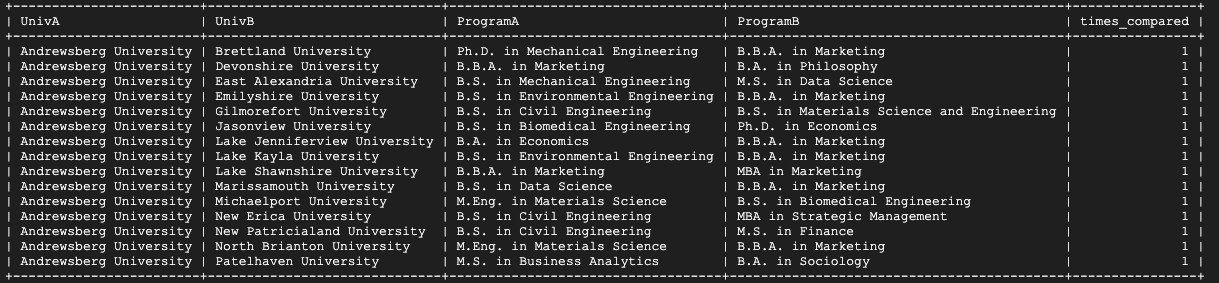
ORDER BY

UnivA,

UnivB,

times\_compared DESC

LIMIT 15;



**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

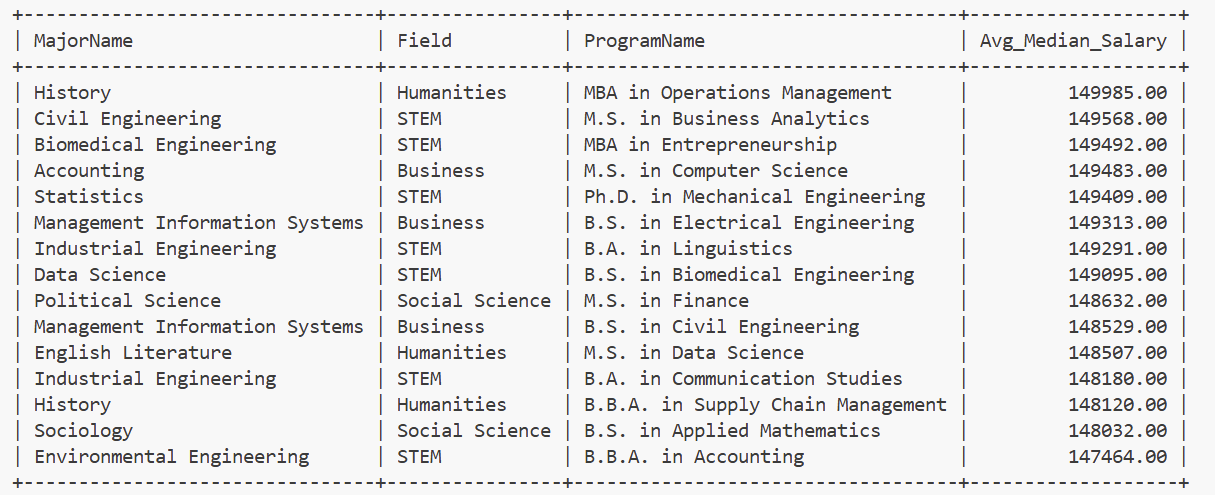
FROM Program p

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;



**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

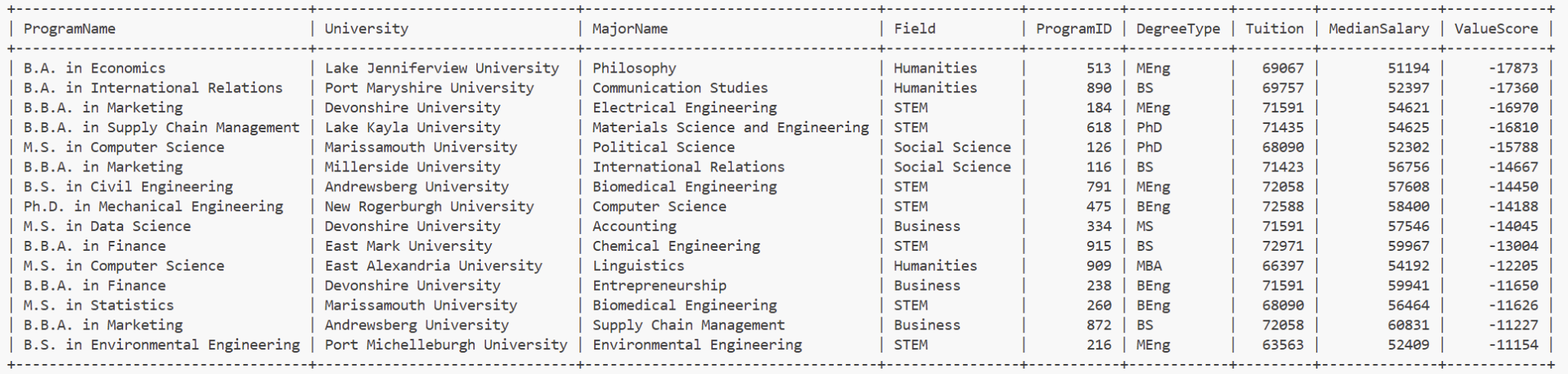
JOIN Major m ON p.MajorID = m.MajorID

WHERE p.MedianSalary IS NOT NULL

AND u.Tuition > (SELECT AVG(Tuition) FROM University)

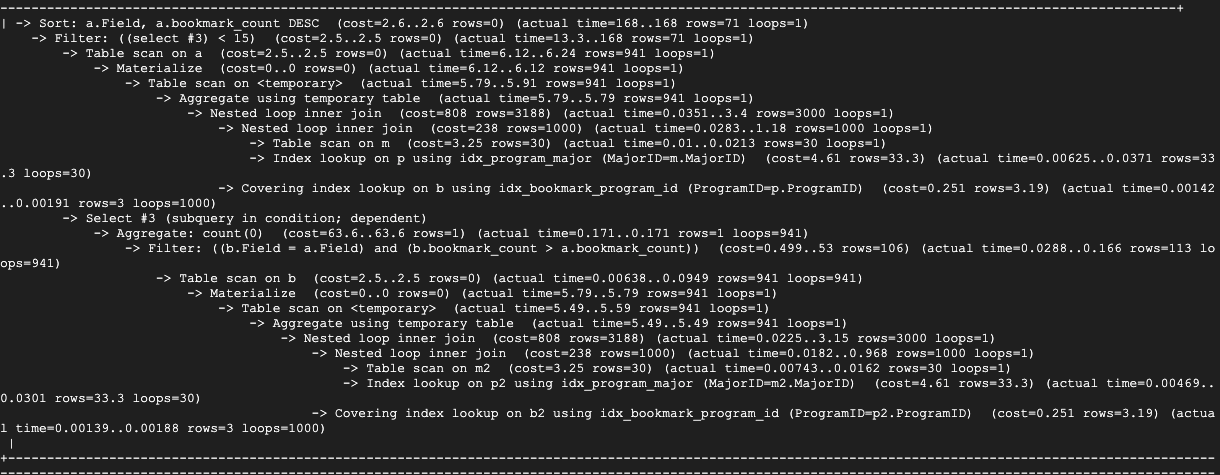
ORDER BY ValueScore ASC

LIMIT 15;

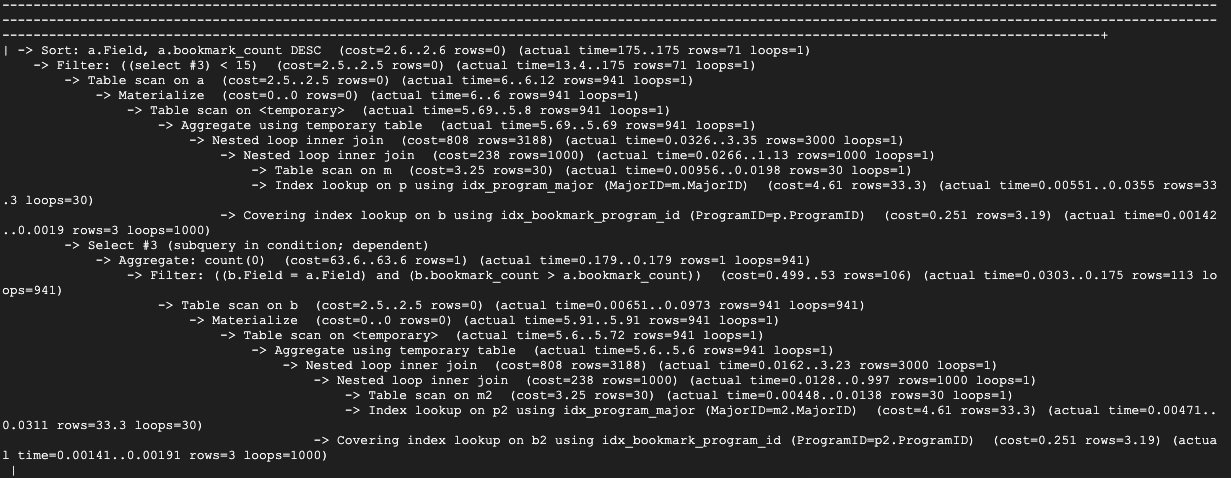


**Analyze Result:**

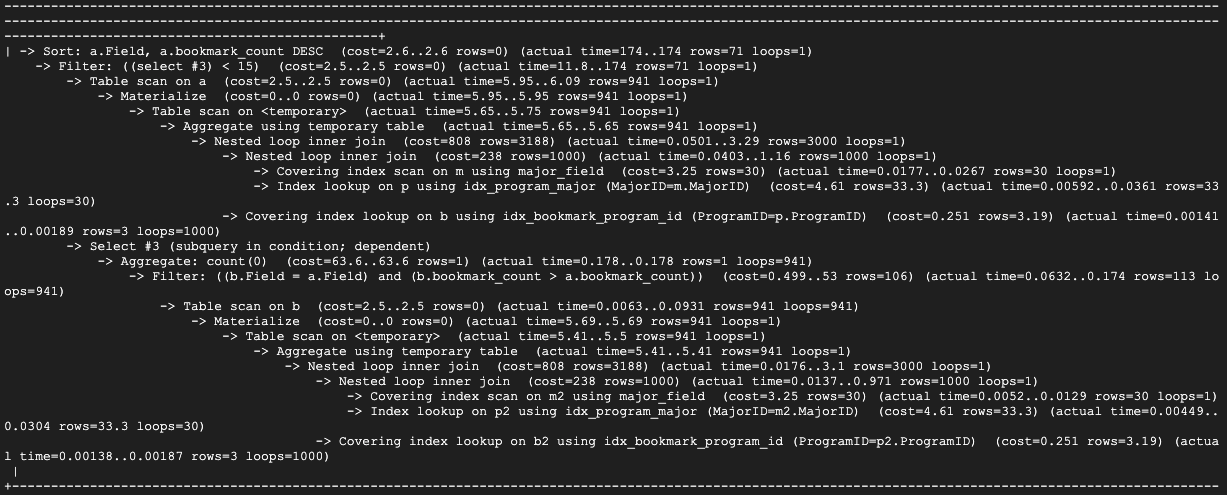
**Query 1 without any index:**

****

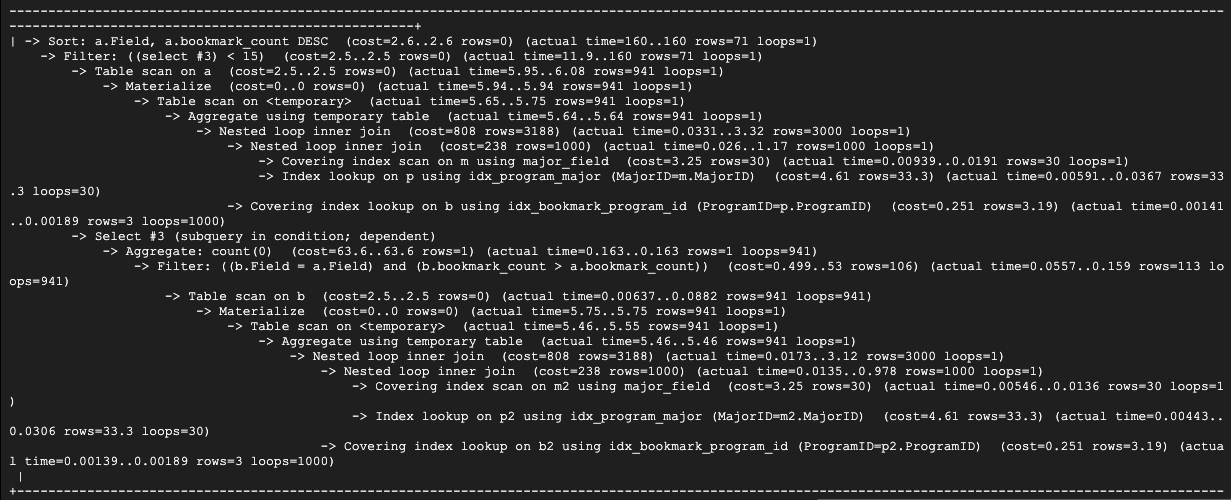
With index on Program.Name:



With index on Major.Field and Program.Name:

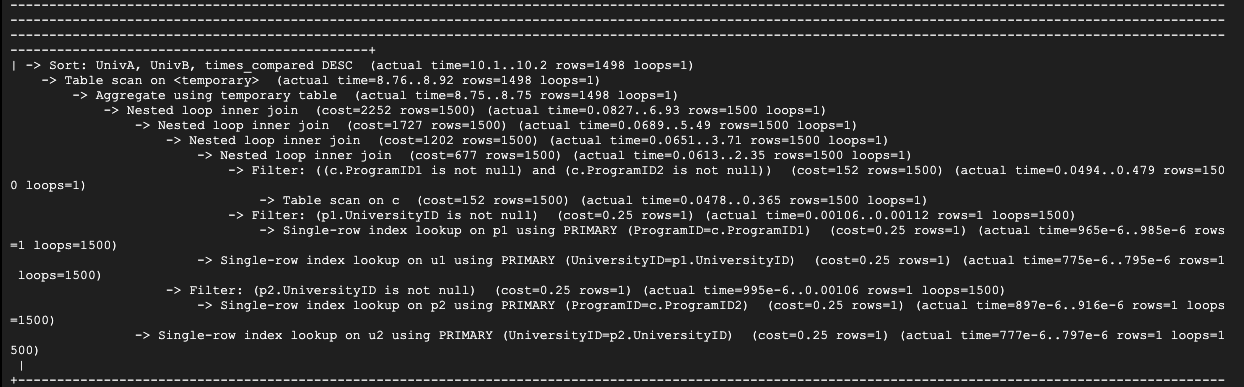


With index on Major.Field:

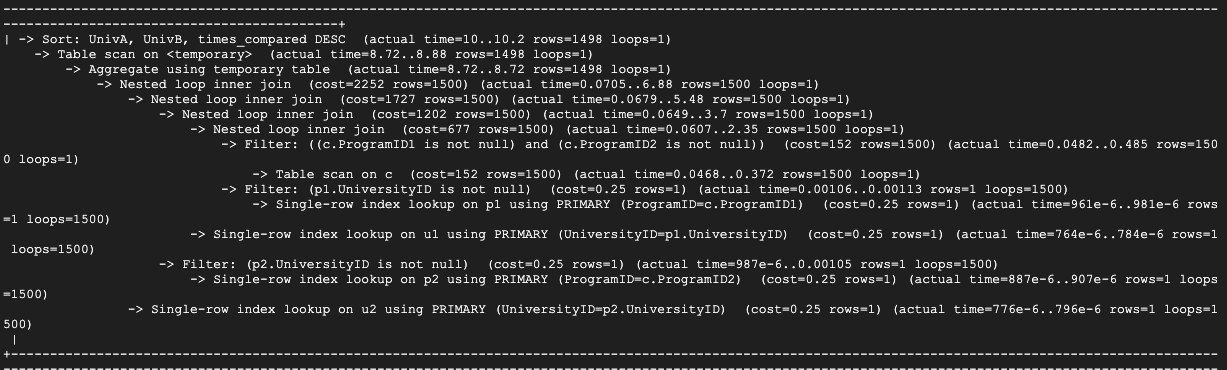


Based on the result shown above, we will keep **only the Major(Field) index**. This is because the query’s ranking logic is per-Field. The Field index makes the repeated “same Field” checks cheaper during the correlated subquery and reduces inner join work, producing a consistent runtime reduction in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

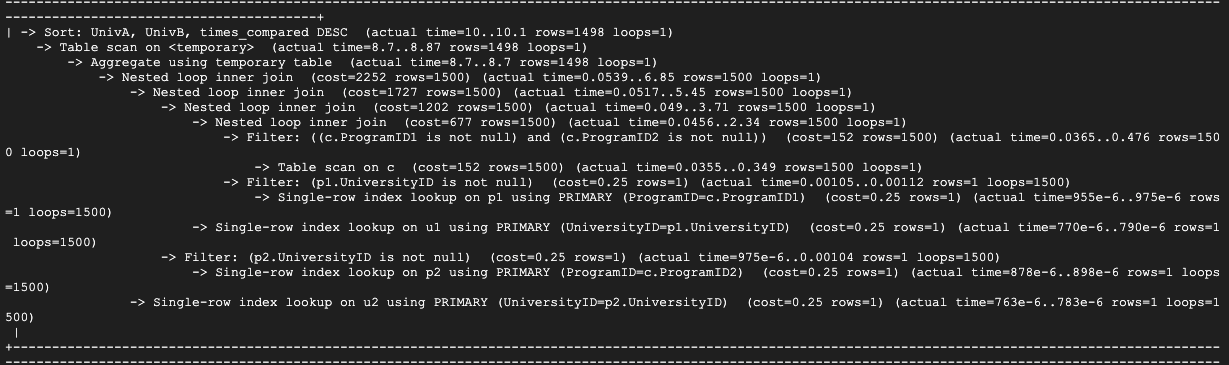
**Query 2 without index:**



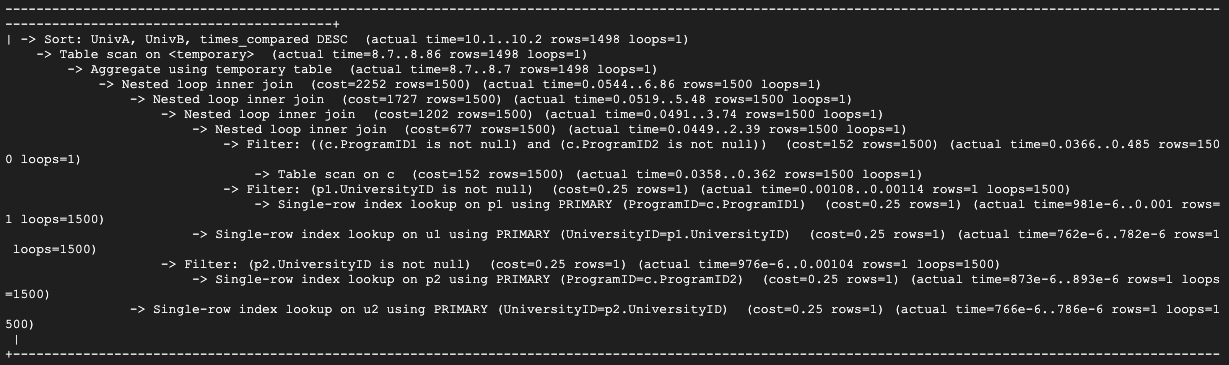
With index on University.Name:



With index on Program.Name:

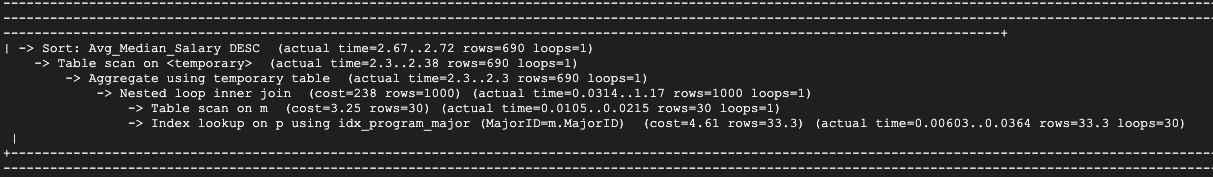


With index on Program.Name and University.Name:

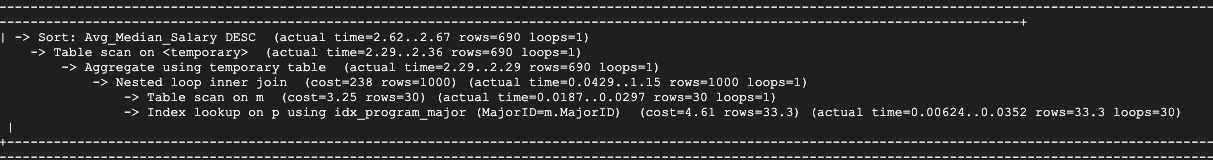


Based on the result shown above, we will **not keep any secondary indexes for this query**. EXPLAIN ANALYZE with University(Name),Program(Name), and both together shows **no change** in the dominant operators-the Aggregate using temporary table and the final Sort(times\_compared DESC) have essentially 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.

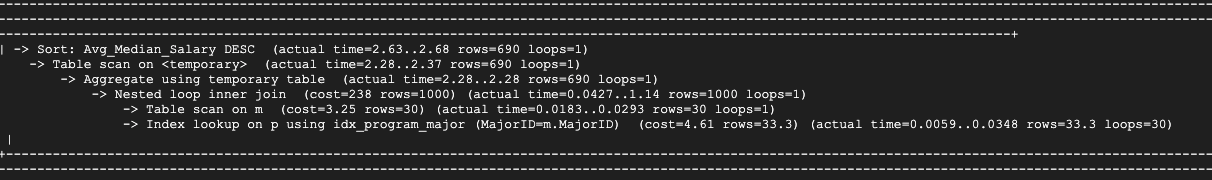
**Query 3 without index:**



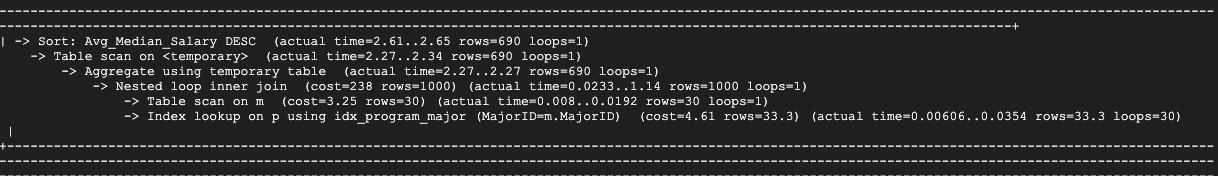
With index on Program.Mediansalary:



With index on Major.MajorName:

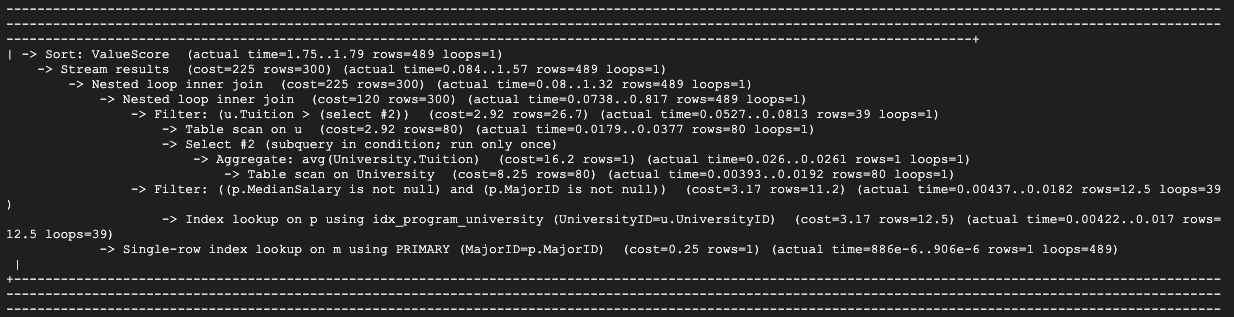


With index on Major.MajorName and Program.MedianSalary:

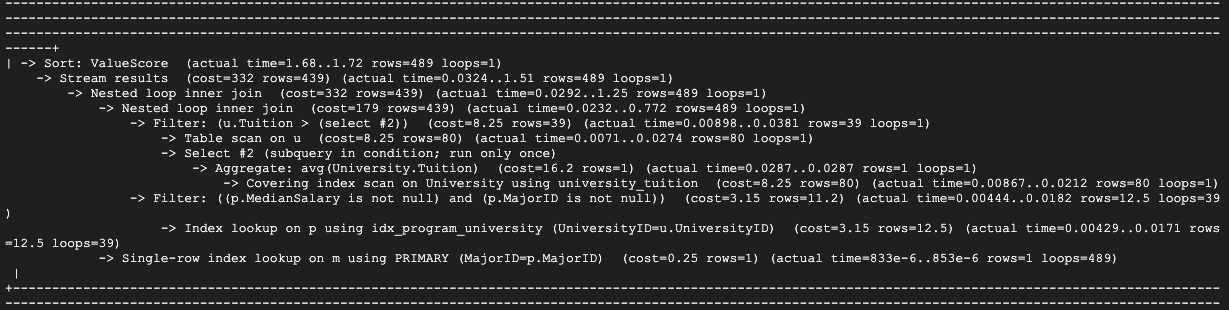


Based on the result shown above, we will keep the index **Program(MedianSalary) and Major(MajorName)**.Since The MedianSalary index will narrow I/O for the AVG aggregation, lowering the “Aggregate using temporary table” time, while the MajorName speeds the join of grouped keys. In combination they deliver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

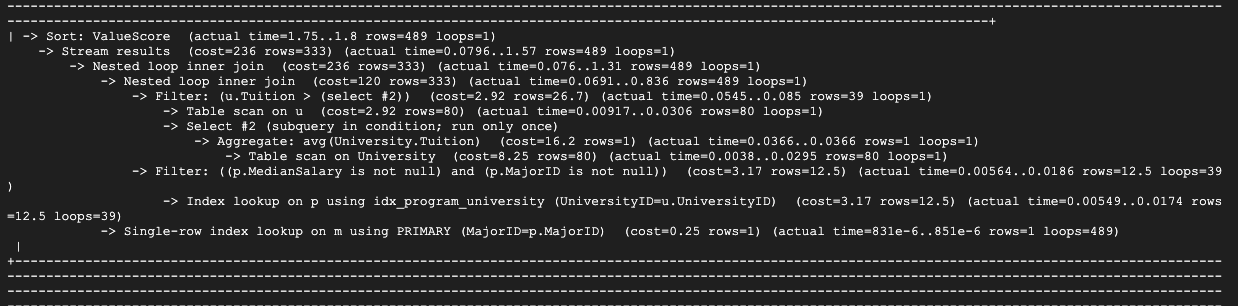
**Query 4 without index:**



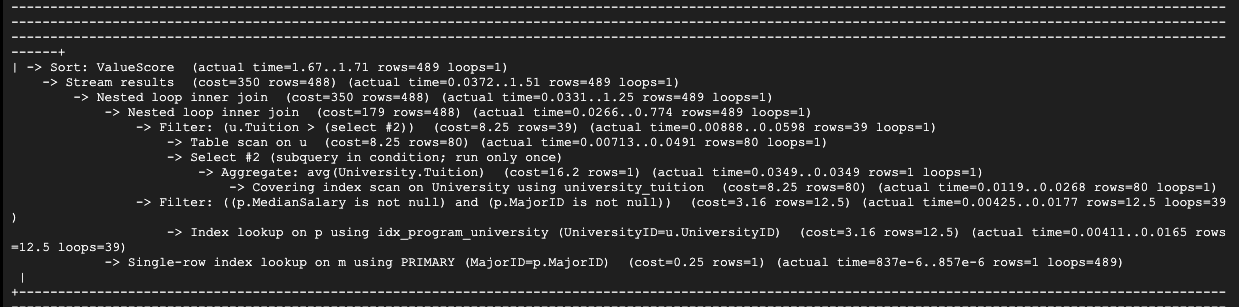
With index on University.tuition:



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.