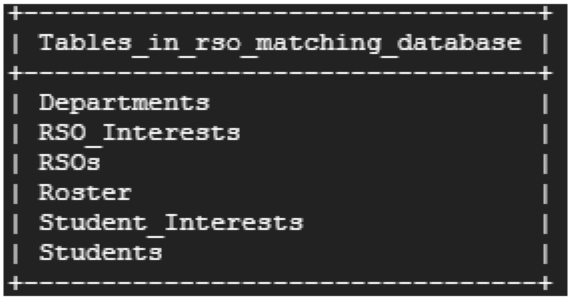
Stage 3 Submission

Setting Up the Tables:

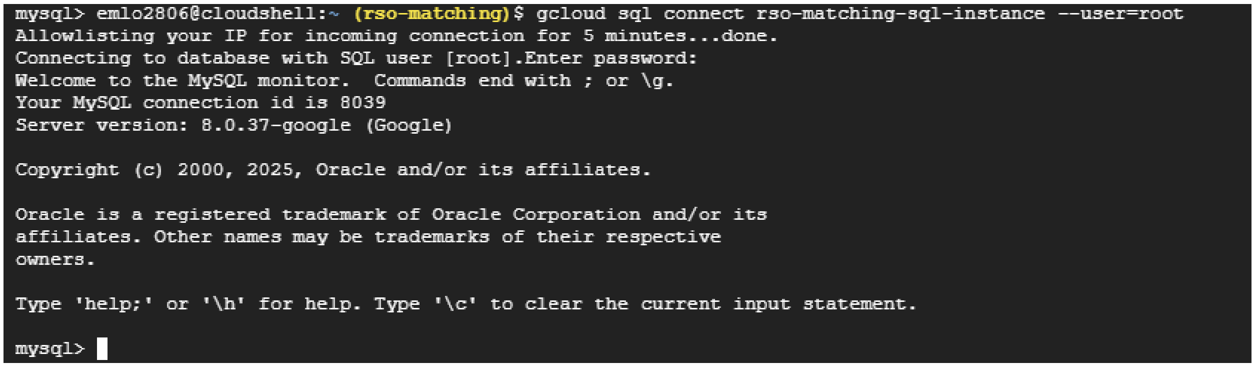
```
CREATE TABLE Students (
    netId VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100),
    year INT,
    minor VARCHAR(50),
    major VARCHAR(50),
    taggedPref VARCHAR(50),
    prefTimeComm VARCHAR(100)
);

CREATE TABLE Student_Interests (
    studentInterestId VARCHAR(50) PRIMARY KEY,
    netId VARCHAR(50),
    interest1 VARCHAR(100),
    interest2 VARCHAR(100),
    interest3 VARCHAR(100),
    FOREIGN KEY (netId) REFERENCES Students(netId)
);

CREATE TABLE Departments (
    departmentName VARCHAR(50) PRIMARY KEY,
    mainOfficeLocation VARCHAR(100),
    numberOfStudents INT
);
```

```
CREATE TABLE RSOs (
    RSOName VARCHAR(100) PRIMARY KEY,
    department VARCHAR(100),
    expTimeComm VARCHAR(100),
    taggedPref VARCHAR(50),
    FOREIGN KEY (department) REFERENCES Departments(departmentName)
);

CREATE TABLE Roster (
    netId VARCHAR(50),
    RSO_name VARCHAR(100),
    PRIMARY KEY (netId, RSO_name),
    FOREIGN KEY (netId) REFERENCES Students(netId),
    FOREIGN KEY (RSO_name) REFERENCES RSOs(RSOName)
);

CREATE TABLE RSO_Interests (
    RSOInterestId VARCHAR(50),
    RSOname VARCHAR(100) PRIMARY KEY,
    RSOInterest1 VARCHAR(100),
    RSOInterest2 VARCHAR(100),
    RSOInterest3 VARCHAR(100),
    FOREIGN KEY (RSOname) REFERENCES RSOs(RSOName)
);
```



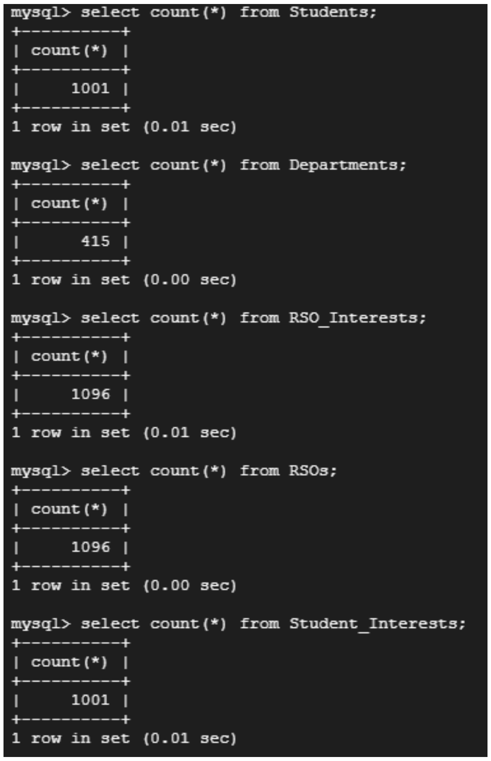
```
+--------------------------------+
| Tables_in_rso_matching_database |
+--------------------------------+
| Departments                    |
| RSO_Interests                  |
| RSOs                           |
| Roster                         |
| Student_Interests              |
| Students                       |
+--------------------------------+
```

Proof of Connection:

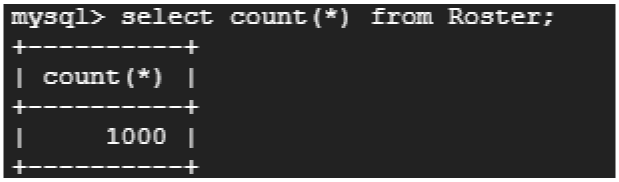


```
mysql> emlo2806@cloudshell:~ (rso-matching)$ gcloud sql connect rso-matching-sql-instance --user=root
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8039
Server version: 8.0.37-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> 
```

Table Sizes:



```
mysql> select count(*) from Students;
+----------+
| count(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from Departments;
+----------+
| count(*) |
+----------+
|      415 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from RSO_Interests;
+----------+
| count(*) |
+----------+
|     1096 |
+----------+
1 row in set (0.01 sec)

mysql> select count(*) from RSOs;
+----------+
| count(*) |
+----------+
|     1096 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from Student_Interests;
+----------+
| count(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.01 sec)
```

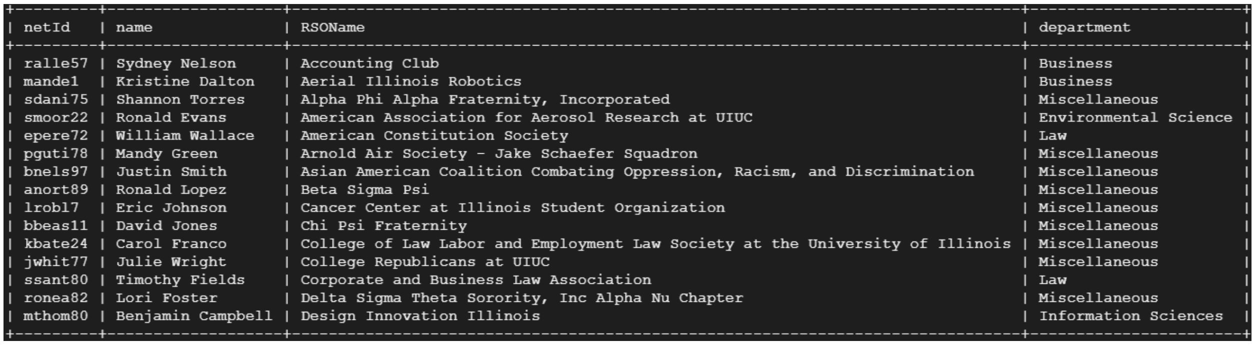


```
mysql> select count(*) from Roster;
+----------+
| count(*) |
+----------+
|     1000 |
+----------+
```

4 Advanced Queries:

1)Purpose: Recommend RSOs to students based on shared interests.

```
SELECT s.netId, s.name, r.RSOName, r.department
FROM Students s
JOIN Roster rs ON s.netId = rs.netId
JOIN RSOs r ON rs.RSO_name = r.RSOName
WHERE r.RSOName IN (
   SELECT ri.RSOname
   FROM RSO_Interests ri
   JOIN Student_Interests si ON
      (ri.RSOInterest1 = si.interest1 OR
       ri.RSOInterest1 = si.interest2 OR
       ri.RSOInterest1 = si.interest3)
   WHERE si.netId = s.netId
);
```

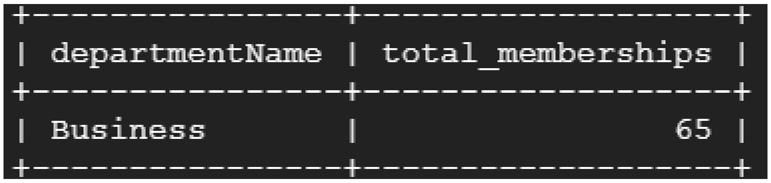


```
+---------+-------------------+-------------------------------------------------------------------------------+-----------------------+
| netId   | name              | RSOName                                                                       | department            |
+---------+-------------------+-------------------------------------------------------------------------------+-----------------------+
| ralle57 | Sydney Nelson     | Accounting Club                                                               | Business              |
| mande1  | Kristine Dalton   | Aerial Illinois Robotics                                                      | Business              |
| sdani75 | Shannon Torres    | Alpha Phi Alpha Fraternity, Incorporated                                      | Miscellaneous         |
| smoor22 | Ronald Evans      | American Association for Aerosol Research at UIUC                             | Environmental Science |
| epere72 | William Wallace   | American Constitution Society                                                 | Law                   |
| pguti78 | Mandy Green       | Arnold Air Society - Jake Schaefer Squadron                                   | Miscellaneous         |
| bnels97 | Justin Smith      | Asian American Coalition Combating Oppression, Racism, and Discrimination     | Miscellaneous         |
| anort89 | Ronald Lopez      | Beta Sigma Psi                                                                | Miscellaneous         |
| lrobl7  | Eric Johnson      | Cancer Center at Illinois Student Organization                                | Miscellaneous         |
| bbeas11 | David Jones       | Chi Psi Fraternity                                                            | Miscellaneous         |
| kbate24 | Carol Franco      | College of Law Labor and Employment Law Society at the University of Illinois | Miscellaneous         |
| jwhit77 | Julie Wright      | College Republicans at UIUC                                                   | Miscellaneous         |
| ssant80 | Timothy Fields    | Corporate and Business Law Association                                        | Law                   |
| ronea82 | Lori Foster       | Delta Sigma Theta Sorority, Inc Alpha Nu Chapter                              | Miscellaneous         |
| mthom80 | Benjamin Campbell | Design Innovation Illinois                                                    | Information Sciences  |
+---------+-------------------+-------------------------------------------------------------------------------+-----------------------+
```

2) Purpose: Identify departments where students join the most RSOs.

```
SELECT d.departmentName, COUNT(r.netId) AS total_memberships
FROM Departments d
JOIN Students s ON d.departmentName = s.major
JOIN Roster r ON s.netId = r.netId
GROUP BY d.departmentName
HAVING COUNT(r.netId) > 5  -- Only departments with >5 memberships
ORDER BY total_memberships DESC
LIMIT 15;
```

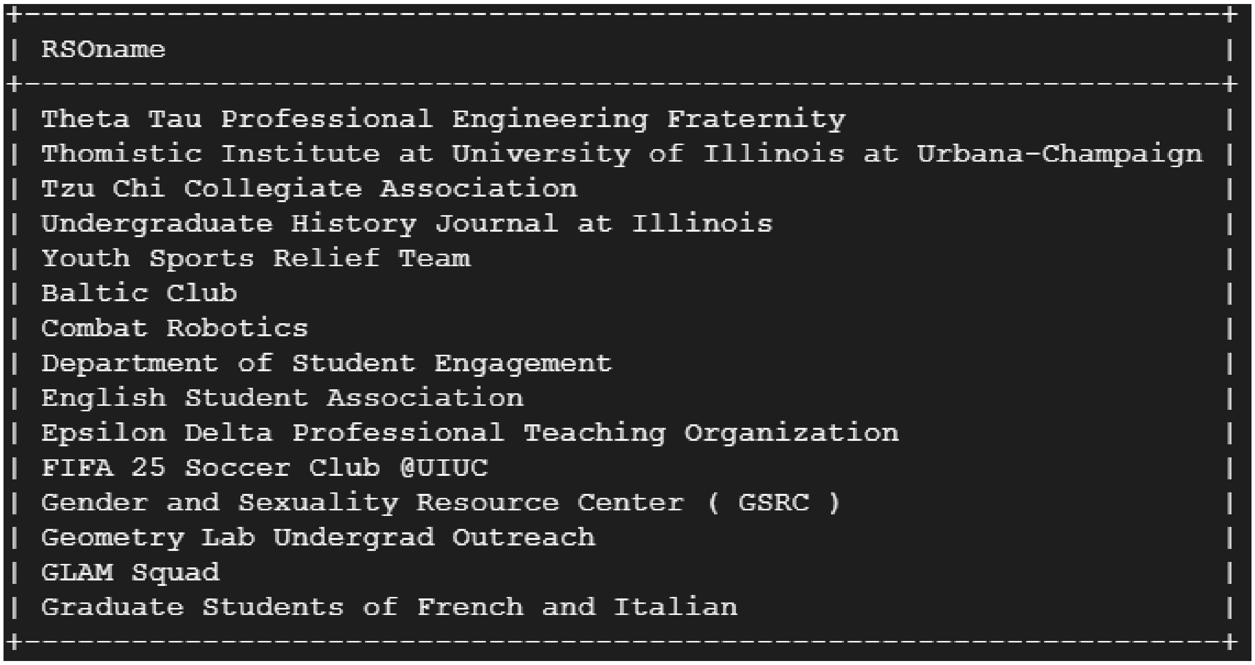
SHOULD ONLY HAVE ONE OUTPUT BECAUSE IT IS THE MOST JOINED DEPARTMENT!!



```
+----------------+-------------------+
| departmentName | total_memberships |
+----------------+-------------------+
| Business       |                65 |
+----------------+-------------------+
```

3) Purpose: Highlight RSOs with niche interests.

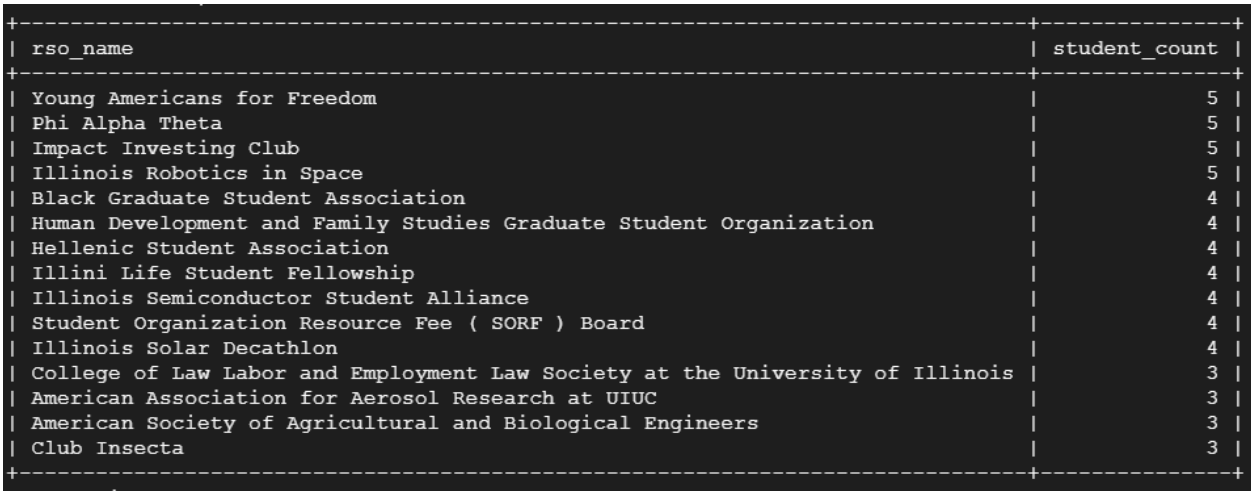
```
SELECT DISTINCT ri.RSOname
FROM RSO_Interests ri
WHERE ri.RSOInterest1 NOT IN (
   SELECT interest1 FROM Student_Interests
   UNION
   SELECT interest2 FROM Student_Interests
   UNION
   SELECT interest3 FROM Student_Interests
)
LIMIT 15;
```



```
+--------------------------------------------------------------------+
| RSOname                                                            |
+--------------------------------------------------------------------+
| Theta Tau Professional Engineering Fraternity                      |
| Thomistic Institute at University of Illinois at Urbana-Champaign |
| Tzu Chi Collegiate Association                                     |
| Undergraduate History Journal at Illinois                          |
| Youth Sports Relief Team                                           |
| Baltic Club                                                        |
| Combat Robotics                                                    |
| Department of Student Engagement                                   |
| English Student Association                                        |
| Epsilon Delta Professional Teaching Organization                   |
| FIFA 25 Soccer Club @UIUC                                          |
| Gender and Sexuality Resource Center ( GSRC )                      |
| Geometry Lab Undergrad Outreach                                    |
| GLAM Squad                                                         |
| Graduate Students of French and Italian                            |
+--------------------------------------------------------------------+
```

4) Purpose: Show how many students are in each RSO (basic analytics for membership tracking).

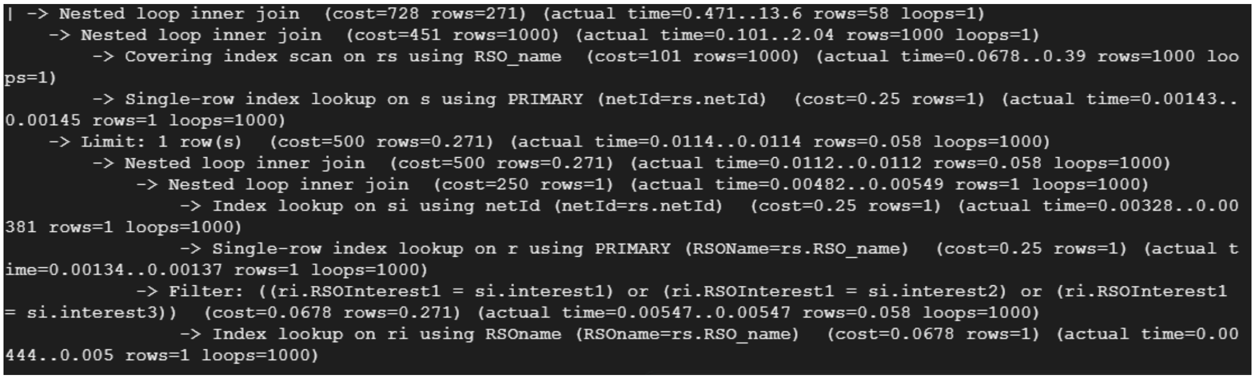
SELECT
    r.RSOName AS rso_name,
    COUNT(ro.netId) AS student_count
FROM RSOs r
LEFT JOIN Roster ro ON r.RSOName = ro.RSO_name
GROUP BY r.RSOName
ORDER BY student_count DESC
LIMIT 15;



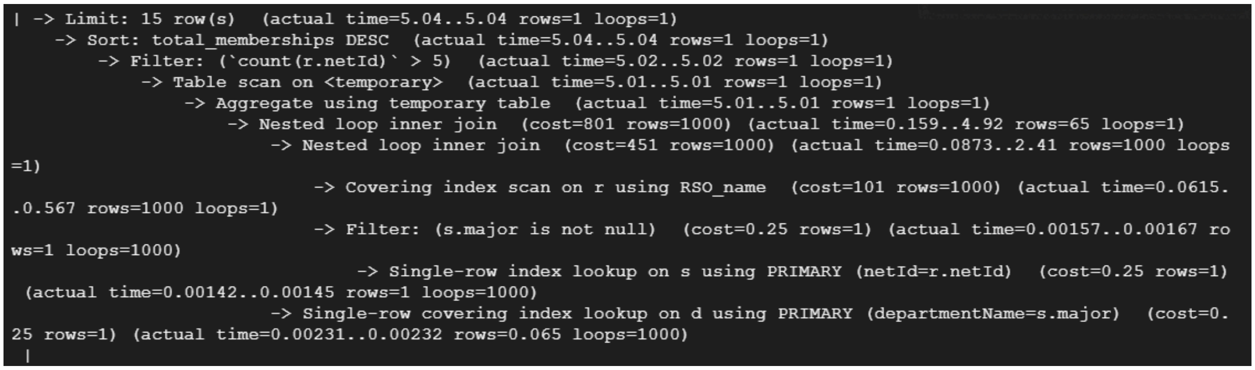
```
+-------------------------------------------------------------------------------+---------------+
| rso_name                                                                      | student_count |
+-------------------------------------------------------------------------------+---------------+
| Young Americans for Freedom                                                   |             5 |
| Phi Alpha Theta                                                               |             5 |
| Impact Investing Club                                                         |             5 |
| Illinois Robotics in Space                                                    |             5 |
| Black Graduate Student Association                                            |             4 |
| Human Development and Family Studies Graduate Student Organization            |             4 |
| Hellenic Student Association                                                  |             4 |
| Illini Life Student Fellowship                                                |             4 |
| Illinois Semiconductor Student Alliance                                       |             4 |
| Student Organization Resource Fee ( SORF ) Board                              |             4 |
| Illinois Solar Decathlon                                                      |             4 |
| College of Law Labor and Employment Law Society at the University of Illinois |             3 |
| American Association for Aerosol Research at UIUC                             |             3 |
| American Society of Agricultural and Biological Engineers                     |             3 |
| Club Insecta                                                                  |             3 |
+-------------------------------------------------------------------------------+---------------+
```

Explain Analyze For Each Query:

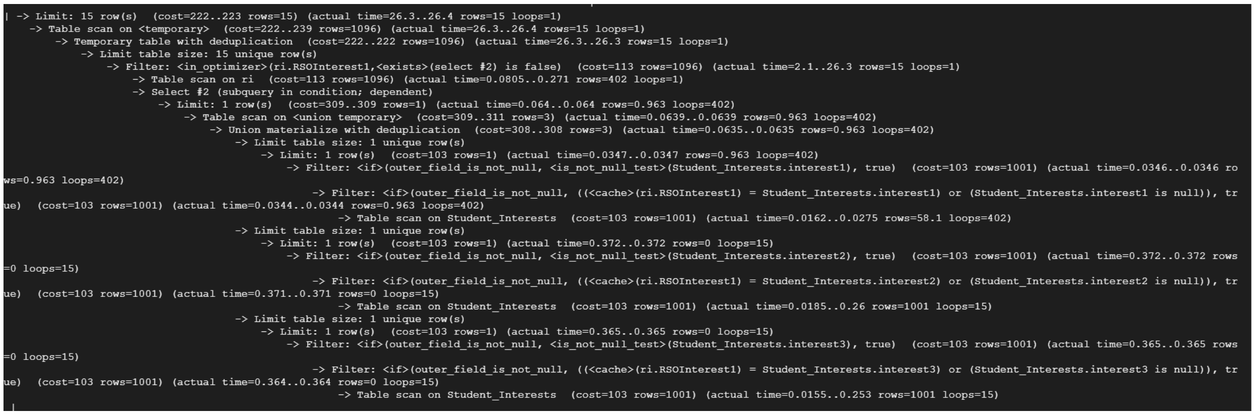
1)



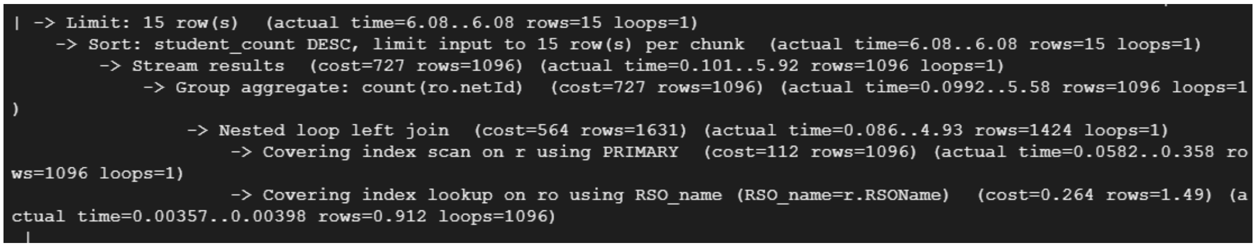
```
| -> Nested loop inner join  (cost=728 rows=271) (actual time=0.471..13.6 rows=58 loops=1)
    -> Nested loop inner join  (cost=451 rows=1000) (actual time=0.101..2.04 rows=1000 loops=1)
        -> Covering index scan on rs using RSO_name  (cost=101 rows=1000) (actual time=0.0678..0.39 rows=1000 loo
ps=1)
        -> Single-row index lookup on s using PRIMARY (netId=rs.netId)  (cost=0.25 rows=1) (actual time=0.00143..
0.00145 rows=1 loops=1000)
    -> Limit: 1 row(s)  (cost=500 rows=0.271) (actual time=0.0114..0.0114 rows=0.058 loops=1000)
        -> Nested loop inner join  (cost=500 rows=0.271) (actual time=0.0112..0.0112 rows=0.058 loops=1000)
            -> Nested loop inner join  (cost=250 rows=1) (actual time=0.00482..0.00549 rows=1 loops=1000)
                -> Index lookup on si using netId (netId=rs.netId)  (cost=0.25 rows=1) (actual time=0.00328..0.00
381 rows=1 loops=1000)
                -> Single-row index lookup on r using PRIMARY (RSOName=rs.RSO_name)  (cost=0.25 rows=1) (actual t
ime=0.00134..0.00137 rows=1 loops=1000)
            -> Filter: ((ri.RSOInterest1 = si.interest1) or (ri.RSOInterest1 = si.interest2) or (ri.RSOInterest1
= si.interest3))  (cost=0.0678 rows=0.271) (actual time=0.00547..0.00547 rows=0.058 loops=1000)
                -> Index lookup on ri using RSOname (RSOname=rs.RSO_name)  (cost=0.0678 rows=1) (actual time=0.00
444..0.005 rows=1 loops=1000)
```

2) 

```
| -> Limit: 15 row(s)  (actual time=5.04..5.04 rows=1 loops=1)
    -> Sort: total_memberships DESC  (actual time=5.04..5.04 rows=1 loops=1)
        -> Filter: (`count(r.netId)` > 5)  (actual time=5.02..5.02 rows=1 loops=1)
            -> Table scan on <temporary>  (actual time=5.01..5.01 rows=1 loops=1)
                -> Aggregate using temporary table  (actual time=5.01..5.01 rows=1 loops=1)
                    -> Nested loop inner join  (cost=801 rows=1000) (actual time=0.159..4.92 rows=65 loops=1)
                        -> Nested loop inner join  (cost=451 rows=1000) (actual time=0.0873..2.41 rows=1000 loops
=1)
                            -> Covering index scan on r using RSO_name  (cost=101 rows=1000) (actual time=0.0615.
.0.567 rows=1000 loops=1)
                            -> Filter: (s.major is not null)  (cost=0.25 rows=1) (actual time=0.00157..0.00167 ro
ws=1 loops=1000)
                                -> Single-row index lookup on s using PRIMARY (netId=r.netId)  (cost=0.25 rows=1)
 (actual time=0.00142..0.00145 rows=1 loops=1000)
                        -> Single-row covering index lookup on d using PRIMARY (departmentName=s.major)  (cost=0.
25 rows=1) (actual time=0.00231..0.00232 rows=0.065 loops=1000)
 |
```

3) 

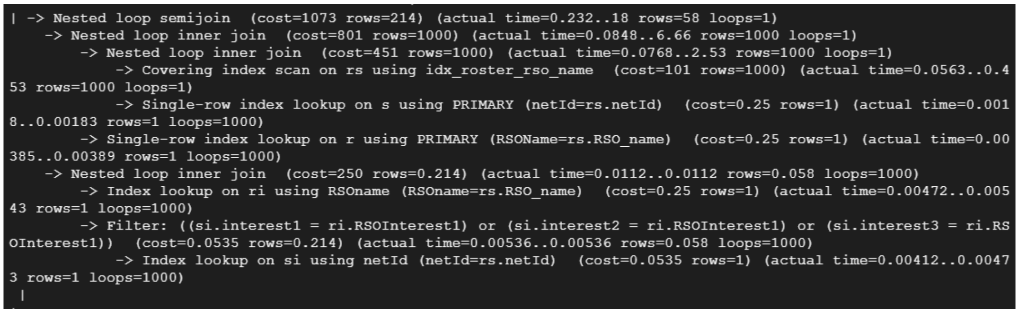
```
| -> Limit: 15 row(s)  (cost=222..223 rows=15) (actual time=26.3..26.4 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=222..239 rows=1096) (actual time=26.3..26.4 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=222..222 rows=1096) (actual time=26.3..26.3 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: <in_optimizer>(ri.RSOInterest1,<exists>(select #2) is false)  (cost=113 rows=1096) (actual time=2.1..26.3 rows=15 loops=1)
                    -> Table scan on ri  (cost=113 rows=1096) (actual time=0.0805..0.271 rows=402 loops=1)
                    -> Select #2 (subquery in condition; dependent)
                        -> Limit: 1 row(s)  (cost=309..309 rows=1) (actual time=0.064..0.064 rows=0.963 loops=402)
                            -> Table scan on <union temporary>  (cost=309..311 rows=3) (actual time=0.0639..0.0639 rows=0.963 loops=402)
                                -> Union materialize with deduplication  (cost=308..308 rows=3) (actual time=0.0635..0.0635 rows=0.963 loops=402)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.0347..0.0347 rows=0.963 loops=402)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest1), true)  (cost=103 rows=1001) (actual time=0.0346..0.0346 ro
ws=0.963 loops=402)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Interests.interest1) or (Student_Interests.interest1 is null)), tr
ue)  (cost=103 rows=1001) (actual time=0.0344..0.0344 rows=0.963 loops=402)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.0162..0.0275 rows=58.1 loops=402)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.372..0.372 rows=0 loops=15)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest2), true)  (cost=103 rows=1001) (actual time=0.372..0.372 rows
=0 loops=15)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Interests.interest2) or (Student_Interests.interest2 is null)), tr
ue)  (cost=103 rows=1001) (actual time=0.371..0.371 rows=0 loops=15)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.0185..0.26 rows=1001 loops=15)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.365..0.365 rows=0 loops=15)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest3), true)  (cost=103 rows=1001) (actual time=0.365..0.365 rows
=0 loops=15)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Interests.interest3) or (Student_Interests.interest3 is null)), tr
ue)  (cost=103 rows=1001) (actual time=0.364..0.364 rows=0 loops=15)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.0155..0.253 rows=1001 loops=15)
 |
```

4) 

```
| -> Limit: 15 row(s)  (actual time=6.08..6.08 rows=15 loops=1)
    -> Sort: student_count DESC, limit input to 15 row(s) per chunk  (actual time=6.08..6.08 rows=15 loops=1)
        -> Stream results  (cost=727 rows=1096) (actual time=0.101..5.92 rows=1096 loops=1)
            -> Group aggregate: count(ro.netId)  (cost=727 rows=1096) (actual time=0.0992..5.58 rows=1096 loops=1
)
                -> Nested loop left join  (cost=564 rows=1631) (actual time=0.086..4.93 rows=1424 loops=1)
                    -> Covering index scan on r using PRIMARY  (cost=112 rows=1096) (actual time=0.0582..0.358 ro
ws=1096 loops=1)
                    -> Covering index lookup on ro using RSO_name (RSO_name=r.RSOName)  (cost=0.264 rows=1.49) (a
ctual time=0.00357..0.00398 rows=0.912 loops=1096)
 |
```

Explore Tradeoffs

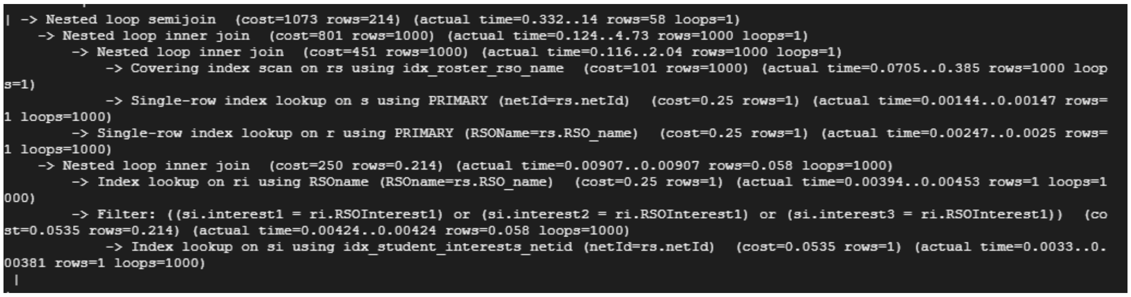
1) Advanced Query 1
    a) CREATE INDEX idx_si_interests ON Student_Interests(interest1, interest2, interest3);



```
| -> Nested loop semijoin  (cost=1073 rows=214) (actual time=0.232..18 rows=58 loops=1)
    -> Nested loop inner join  (cost=801 rows=1000) (actual time=0.0848..6.66 rows=1000 loops=1)
        -> Nested loop inner join  (cost=451 rows=1000) (actual time=0.0768..2.53 rows=1000 loops=1)
            -> Covering index scan on rs using idx_roster_rso_name  (cost=101 rows=1000) (actual time=0.0563..0.4
53 rows=1000 loops=1)
            -> Single-row index lookup on s using PRIMARY (netId=rs.netId)  (cost=0.25 rows=1) (actual time=0.001
8..0.00183 rows=1 loops=1000)
        -> Single-row index lookup on r using PRIMARY (RSOName=rs.RSO_name)  (cost=0.25 rows=1) (actual time=0.00
385..0.00389 rows=1 loops=1000)
    -> Nested loop inner join  (cost=250 rows=0.214) (actual time=0.0112..0.0112 rows=0.058 loops=1000)
        -> Index lookup on ri using RSOname (RSOname=rs.RSO_name)  (cost=0.25 rows=1) (actual time=0.00472..0.005
43 rows=1 loops=1000)
        -> Filter: ((si.interest1 = ri.RSOInterest1) or (si.interest2 = ri.RSOInterest1) or (si.interest3 = ri.RS
OInterest1))  (cost=0.0535 rows=0.214) (actual time=0.00536..0.00536 rows=0.058 loops=1000)
            -> Index lookup on si using netId (netId=rs.netId)  (cost=0.0535 rows=1) (actual time=0.00412..0.0047
3 rows=1 loops=1000)
 |
```

        i) This indexing strategy has a higher cost (1073) compared to the default index (728), meaning it has worse performance and is not an ideal indexing strategy.

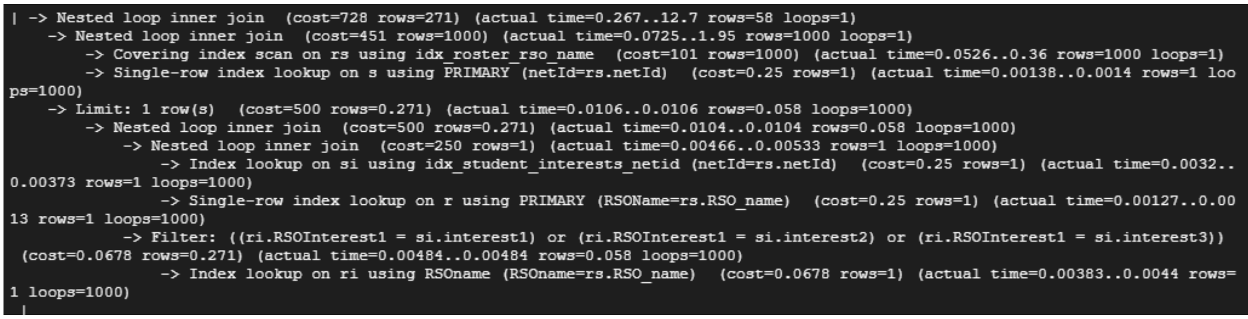
b) CREATE INDEX idx_student_interests_interest1 ON Student_Interests(interest1);



```
| -> Nested loop semijoin  (cost=1073 rows=214) (actual time=0.332..14 rows=58 loops=1)
    -> Nested loop inner join  (cost=801 rows=1000) (actual time=0.124..4.73 rows=1000 loops=1)
        -> Nested loop inner join  (cost=451 rows=1000) (actual time=0.116..2.04 rows=1000 loops=1)
            -> Covering index scan on rs using idx_roster_rso_name  (cost=101 rows=1000) (actual time=0.0705..0.385 rows=1000 loop
s=1)
            -> Single-row index lookup on s using PRIMARY (netId=rs.netId)  (cost=0.25 rows=1) (actual time=0.00144..0.00147 rows=
1 loops=1000)
        -> Single-row index lookup on r using PRIMARY (RSOName=rs.RSO_name)  (cost=0.25 rows=1) (actual time=0.00247..0.0025 rows=
1 loops=1000)
    -> Nested loop inner join  (cost=250 rows=0.214) (actual time=0.00907..0.00907 rows=0.058 loops=1000)
        -> Index lookup on ri using RSOname (RSOname=rs.RSO_name)  (cost=0.25 rows=1) (actual time=0.00394..0.00453 rows=1 loops=1
000)
        -> Filter: ((si.interest1 = ri.RSOInterest1) or (si.interest2 = ri.RSOInterest1) or (si.interest3 = ri.RSOInterest1))  (co
st=0.0535 rows=0.214) (actual time=0.00424..0.00424 rows=0.058 loops=1000)
            -> Index lookup on si using idx_student_interests_netid (netId=rs.netId)  (cost=0.0535 rows=1) (actual time=0.0033..0.
00381 rows=1 loops=1000)
 |
```

i) This indexing strategy has a higher cost (1073) compared to the default index (728), meaning it has worse performance and is not an ideal indexing strategy.

c) CREATE INDEX idx_rso_interests_interest2 ON RSO_Interests(RSOInterest2);

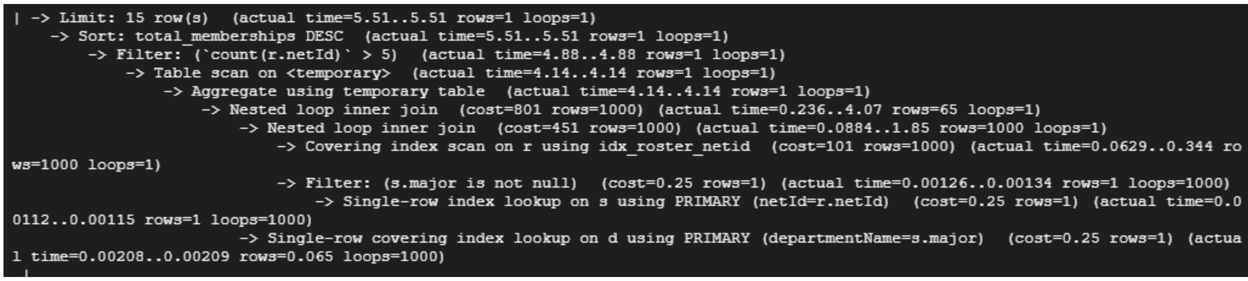


```
| -> Nested loop inner join  (cost=728 rows=271) (actual time=0.267..12.7 rows=58 loops=1)
    -> Nested loop inner join  (cost=451 rows=1000) (actual time=0.0725..1.95 rows=1000 loops=1)
        -> Covering index scan on rs using idx_roster_rso_name  (cost=101 rows=1000) (actual time=0.0526..0.36 rows=1000 loops=1)
        -> Single-row index lookup on s using PRIMARY (netId=rs.netId)  (cost=0.25 rows=1) (actual time=0.00138..0.0014 rows=1 loo
ps=1000)
    -> Limit: 1 row(s)  (cost=500 rows=0.271) (actual time=0.0106..0.0106 rows=0.058 loops=1000)
        -> Nested loop inner join  (cost=500 rows=0.271) (actual time=0.0104..0.0104 rows=0.058 loops=1000)
            -> Nested loop inner join  (cost=250 rows=1) (actual time=0.00466..0.00533 rows=1 loops=1000)
                -> Index lookup on si using idx_student_interests_netid (netId=rs.netId)  (cost=0.25 rows=1) (actual time=0.0032..
0.00373 rows=1 loops=1000)
                -> Single-row index lookup on r using PRIMARY (RSOName=rs.RSO_name)  (cost=0.25 rows=1) (actual time=0.00127..0.00
13 rows=1 loops=1000)
            -> Filter: ((ri.RSOInterest1 = si.interest1) or (ri.RSOInterest1 = si.interest2) or (ri.RSOInterest1 = si.interest3))
 (cost=0.0678 rows=0.271) (actual time=0.00484..0.00484 rows=0.058 loops=1000)
                -> Index lookup on ri using RSOname (RSOname=rs.RSO_name)  (cost=0.0678 rows=1) (actual time=0.00383..0.0044 rows=
1 loops=1000)
 |
```

i) This indexing strategy has the same cost as the default index (728), meaning it doesn’t usefully affect performance.

2) Advanced query 2

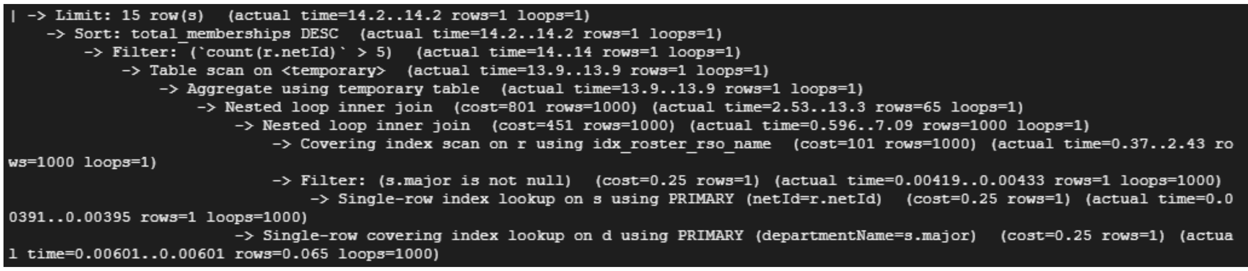
a) CREATE INDEX idx_roster_netid ON Roster(netId);



```
| -> Limit: 15 row(s)  (actual time=5.51..5.51 rows=1 loops=1)
    -> Sort: total_memberships DESC  (actual time=5.51..5.51 rows=1 loops=1)
        -> Filter: (`count(r.netId)` > 5)  (actual time=4.88..4.88 rows=1 loops=1)
            -> Table scan on <temporary>  (actual time=4.14..4.14 rows=1 loops=1)
                -> Aggregate using temporary table  (actual time=4.14..4.14 rows=1 loops=1)
                    -> Nested loop inner join  (cost=801 rows=1000) (actual time=0.236..4.07 rows=65 loops=1)
                        -> Nested loop inner join  (cost=451 rows=1000) (actual time=0.0884..1.85 rows=1000 loops=1)
                            -> Covering index scan on r using idx_roster_netid  (cost=101 rows=1000) (actual time=0.0629..0.344 ro
ws=1000 loops=1)
                            -> Filter: (s.major is not null)  (cost=0.25 rows=1) (actual time=0.00126..0.00134 rows=1 loops=1000)
                                -> Single-row index lookup on s using PRIMARY (netId=r.netId)  (cost=0.25 rows=1) (actual time=0.0
0112..0.00115 rows=1 loops=1000)
                        -> Single-row covering index lookup on d using PRIMARY (departmentName=s.major)  (cost=0.25 rows=1) (actua
l time=0.00208..0.00209 rows=0.065 loops=1000)
```

i) This indexing strategy has the same cost as the default index (801), meaning it doesn’t usefully affect performance.

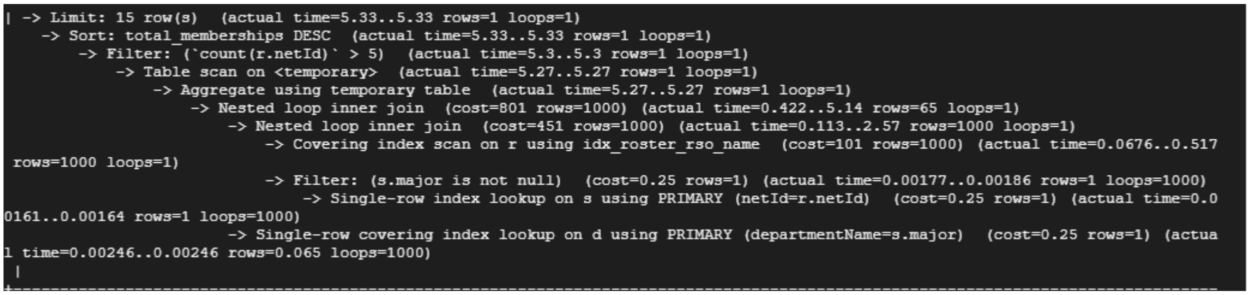
b) CREATE INDEX idx_students_major_netid ON Students(major, netId);



```
| -> Limit: 15 row(s)  (actual time=14.2..14.2 rows=1 loops=1)
    -> Sort: total_memberships DESC  (actual time=14.2..14.2 rows=1 loops=1)
        -> Filter: (`count(r.netId)` > 5)  (actual time=14..14 rows=1 loops=1)
            -> Table scan on <temporary>  (actual time=13.9..13.9 rows=1 loops=1)
                -> Aggregate using temporary table  (actual time=13.9..13.9 rows=1 loops=1)
                    -> Nested loop inner join  (cost=801 rows=1000) (actual time=2.53..13.3 rows=65 loops=1)
                        -> Nested loop inner join  (cost=451 rows=1000) (actual time=0.596..7.09 rows=1000 loops=1)
                            -> Covering index scan on r using idx_roster_rso_name  (cost=101 rows=1000) (actual time=0.37..2.43 ro
ws=1000 loops=1)
                            -> Filter: (s.major is not null)  (cost=0.25 rows=1) (actual time=0.00419..0.00433 rows=1 loops=1000)
                                -> Single-row index lookup on s using PRIMARY (netId=r.netId)  (cost=0.25 rows=1) (actual time=0.0
0391..0.00395 rows=1 loops=1000)
                        -> Single-row covering index lookup on d using PRIMARY (departmentName=s.major)  (cost=0.25 rows=1) (actua
l time=0.00601..0.00601 rows=0.065 loops=1000)
```

i) This indexing strategy has the same cost as the default index (801), meaning it doesn’t usefully affect performance.

c) CREATE INDEX idx_students_major ON Students(major);

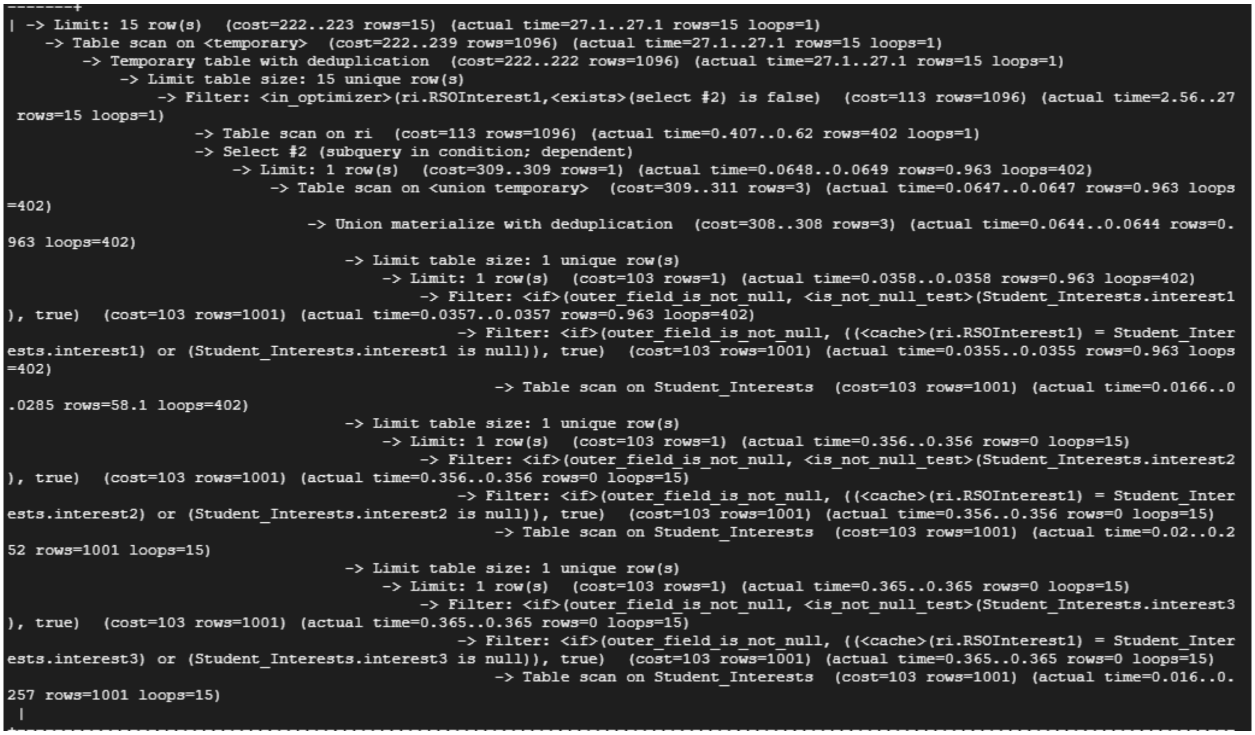


```
| -> Limit: 15 row(s)  (actual time=5.33..5.33 rows=1 loops=1)
    -> Sort: total_memberships DESC  (actual time=5.33..5.33 rows=1 loops=1)
        -> Filter: (`count(r.netId)` > 5)  (actual time=5.3..5.3 rows=1 loops=1)
            -> Table scan on <temporary>  (actual time=5.27..5.27 rows=1 loops=1)
                -> Aggregate using temporary table  (actual time=5.27..5.27 rows=1 loops=1)
                    -> Nested loop inner join  (cost=801 rows=1000) (actual time=0.422..5.14 rows=65 loops=1)
                        -> Nested loop inner join  (cost=451 rows=1000) (actual time=0.113..2.57 rows=1000 loops=1)
                            -> Covering index scan on r using idx_roster_rso_name  (cost=101 rows=1000) (actual time=0.0676..0.517
 rows=1000 loops=1)
                            -> Filter: (s.major is not null)  (cost=0.25 rows=1) (actual time=0.00177..0.00186 rows=1 loops=1000)
                                -> Single-row index lookup on s using PRIMARY (netId=r.netId)  (cost=0.25 rows=1) (actual time=0.0
0161..0.00164 rows=1 loops=1000)
                        -> Single-row covering index lookup on d using PRIMARY (departmentName=s.major)  (cost=0.25 rows=1) (actua
l time=0.00246..0.00246 rows=0.065 loops=1000)
 |
+--------------------------------------------------------------------------------------------------------------------------------
```

i) This indexing strategy has the same cost as the default index (801), meaning it doesn’t usefully affect performance.

3) Advanced query 3

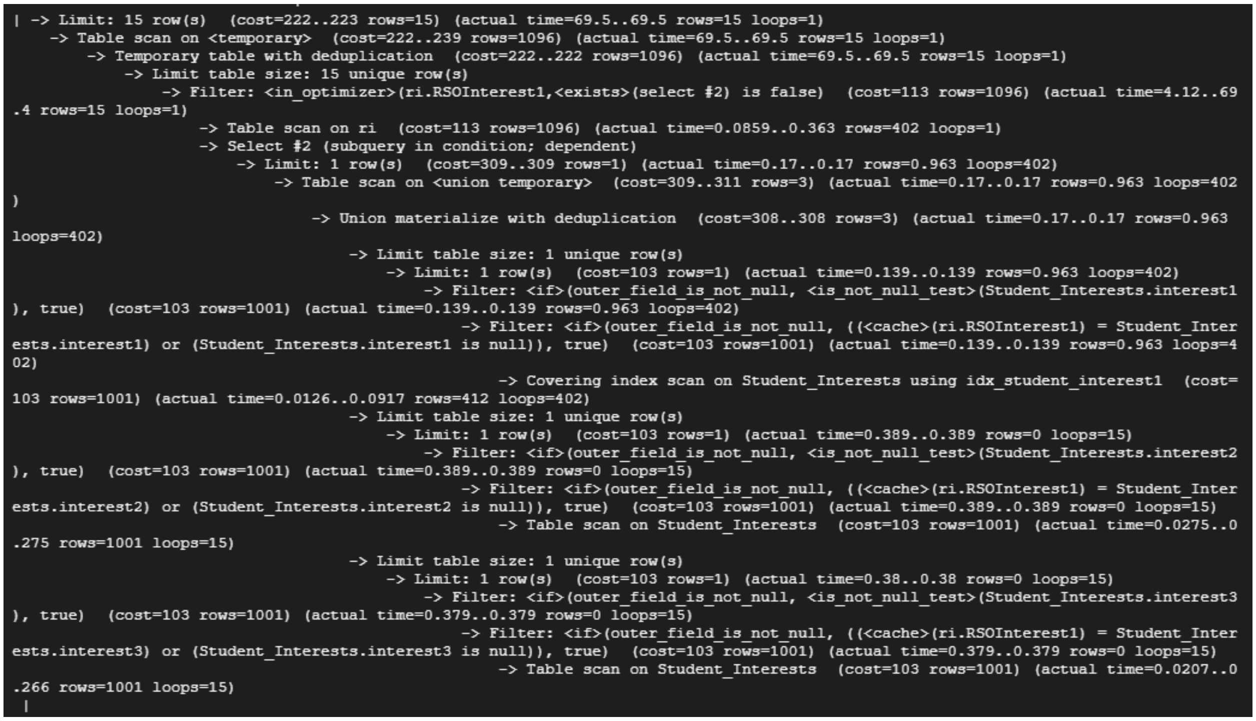
a) CREATE INDEX idx_rso_interest1 ON RSO_Interests(RSOInterest1);



```
--------+
| -> Limit: 15 row(s)  (cost=222..223 rows=15) (actual time=27.1..27.1 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=222..239 rows=1096) (actual time=27.1..27.1 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=222..222 rows=1096) (actual time=27.1..27.1 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: <in_optimizer>(ri.RSOInterest1,<exists>(select #2) is false)  (cost=113 rows=1096) (actual time=2.56..27
 rows=15 loops=1)
                    -> Table scan on ri  (cost=113 rows=1096) (actual time=0.407..0.62 rows=402 loops=1)
                    -> Select #2 (subquery in condition; dependent)
                        -> Limit: 1 row(s)  (cost=309..309 rows=1) (actual time=0.0648..0.0649 rows=0.963 loops=402)
                            -> Table scan on <union temporary>  (cost=309..311 rows=3) (actual time=0.0647..0.0647 rows=0.963 loops
=402)
                                -> Union materialize with deduplication  (cost=308..308 rows=3) (actual time=0.0644..0.0644 rows=0.
963 loops=402)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.0358..0.0358 rows=0.963 loops=402)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest1
), true)  (cost=103 rows=1001) (actual time=0.0357..0.0357 rows=0.963 loops=402)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest1) or (Student_Interests.interest1 is null)), true)  (cost=103 rows=1001) (actual time=0.0355..0.0355 rows=0.963 loops
=402)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.0166..0
.0285 rows=58.1 loops=402)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.356..0.356 rows=0 loops=15)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest2
), true)  (cost=103 rows=1001) (actual time=0.356..0.356 rows=0 loops=15)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest2) or (Student_Interests.interest2 is null)), true)  (cost=103 rows=1001) (actual time=0.356..0.356 rows=0 loops=15)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.02..0.2
52 rows=1001 loops=15)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.365..0.365 rows=0 loops=15)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest3
), true)  (cost=103 rows=1001) (actual time=0.365..0.365 rows=0 loops=15)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest3) or (Student_Interests.interest3 is null)), true)  (cost=103 rows=1001) (actual time=0.365..0.365 rows=0 loops=15)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.016..0.
257 rows=1001 loops=15)
 |
```

i) This indexing strategy has the same cost as the default index (222), meaning it doesn’t usefully affect performance.

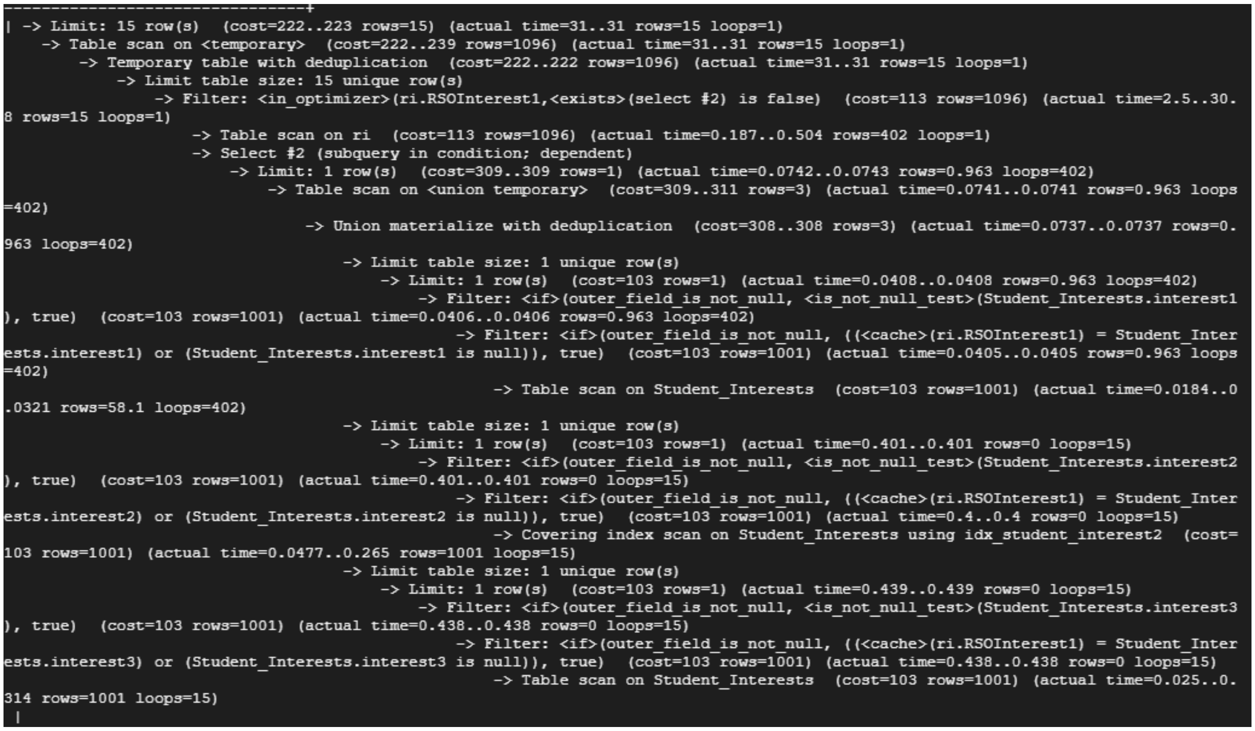
b) CREATE INDEX idx_student_interest1 ON Student_Interests(interest1);



```
| -> Limit: 15 row(s)  (cost=222..223 rows=15) (actual time=69.5..69.5 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=222..239 rows=1096) (actual time=69.5..69.5 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=222..222 rows=1096) (actual time=69.5..69.5 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: <in_optimizer>(ri.RSOInterest1,<exists>(select #2) is false)  (cost=113 rows=1096) (actual time=4.12..69
.4 rows=15 loops=1)
                    -> Table scan on ri  (cost=113 rows=1096) (actual time=0.0859..0.363 rows=402 loops=1)
                    -> Select #2 (subquery in condition; dependent)
                        -> Limit: 1 row(s)  (cost=309..309 rows=1) (actual time=0.17..0.17 rows=0.963 loops=402)
                            -> Table scan on <union temporary>  (cost=309..311 rows=3) (actual time=0.17..0.17 rows=0.963 loops=402
)
                                -> Union materialize with deduplication  (cost=308..308 rows=3) (actual time=0.17..0.17 rows=0.963
loops=402)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.139..0.139 rows=0.963 loops=402)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest1
), true)  (cost=103 rows=1001) (actual time=0.139..0.139 rows=0.963 loops=402)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest1) or (Student_Interests.interest1 is null)), true)  (cost=103 rows=1001) (actual time=0.139..0.139 rows=0.963 loops=4
02)
                                                    -> Covering index scan on Student_Interests using idx_student_interest1  (cost=
103 rows=1001) (actual time=0.0126..0.0917 rows=412 loops=402)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.389..0.389 rows=0 loops=15)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest2
), true)  (cost=103 rows=1001) (actual time=0.389..0.389 rows=0 loops=15)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest2) or (Student_Interests.interest2 is null)), true)  (cost=103 rows=1001) (actual time=0.389..0.389 rows=0 loops=15)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.0275..0
.275 rows=1001 loops=15)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.38..0.38 rows=0 loops=15)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest3
), true)  (cost=103 rows=1001) (actual time=0.379..0.379 rows=0 loops=15)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest3) or (Student_Interests.interest3 is null)), true)  (cost=103 rows=1001) (actual time=0.379..0.379 rows=0 loops=15)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.0207..0
.266 rows=1001 loops=15)
 |
```

i) This indexing strategy has the same cost as the default index (222), meaning it doesn't usefully affect performance.

c) CREATE INDEX idx_student_interest2 ON Student_Interests(interest2);

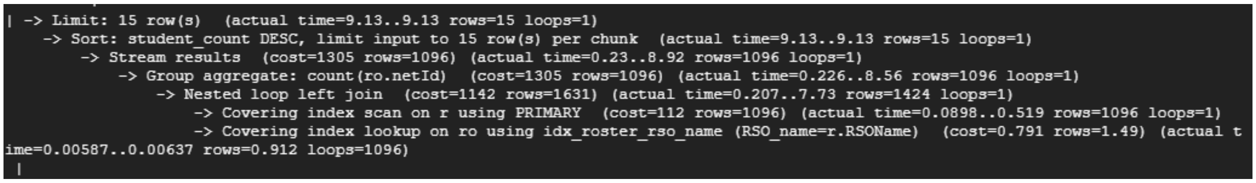


```
-------------------------------+
| -> Limit: 15 row(s)  (cost=222..223 rows=15) (actual time=31..31 rows=15 loops=1)
    -> Table scan on <temporary>  (cost=222..239 rows=1096) (actual time=31..31 rows=15 loops=1)
        -> Temporary table with deduplication  (cost=222..222 rows=1096) (actual time=31..31 rows=15 loops=1)
            -> Limit table size: 15 unique row(s)
                -> Filter: <in_optimizer>(ri.RSOInterest1,<exists>(select #2) is false)  (cost=113 rows=1096) (actual time=2.5..30.
8 rows=15 loops=1)
                    -> Table scan on ri  (cost=113 rows=1096) (actual time=0.187..0.504 rows=402 loops=1)
                    -> Select #2 (subquery in condition; dependent)
                        -> Limit: 1 row(s)  (cost=309..309 rows=1) (actual time=0.0742..0.0743 rows=0.963 loops=402)
                            -> Table scan on <union temporary>  (cost=309..311 rows=3) (actual time=0.0741..0.0741 rows=0.963 loops
=402)
                                -> Union materialize with deduplication  (cost=308..308 rows=3) (actual time=0.0737..0.0737 rows=0.
963 loops=402)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.0408..0.0408 rows=0.963 loops=402)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest1
), true)  (cost=103 rows=1001) (actual time=0.0406..0.0406 rows=0.963 loops=402)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest1) or (Student_Interests.interest1 is null)), true)  (cost=103 rows=1001) (actual time=0.0405..0.0405 rows=0.963 loops
=402)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.0184..0
.0321 rows=58.1 loops=402)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.401..0.401 rows=0 loops=15)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest2
), true)  (cost=103 rows=1001) (actual time=0.401..0.401 rows=0 loops=15)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest2) or (Student_Interests.interest2 is null)), true)  (cost=103 rows=1001) (actual time=0.4..0.4 rows=0 loops=15)
                                                    -> Covering index scan on Student_Interests using idx_student_interest2  (cost=
103 rows=1001) (actual time=0.0477..0.265 rows=1001 loops=15)
                                    -> Limit table size: 1 unique row(s)
                                        -> Limit: 1 row(s)  (cost=103 rows=1) (actual time=0.439..0.439 rows=0 loops=15)
                                            -> Filter: <if>(outer_field_is_not_null, <is_not_null_test>(Student_Interests.interest3
), true)  (cost=103 rows=1001) (actual time=0.438..0.438 rows=0 loops=15)
                                                -> Filter: <if>(outer_field_is_not_null, ((<cache>(ri.RSOInterest1) = Student_Inter
ests.interest3) or (Student_Interests.interest3 is null)), true)  (cost=103 rows=1001) (actual time=0.438..0.438 rows=0 loops=15)
                                                    -> Table scan on Student_Interests  (cost=103 rows=1001) (actual time=0.025..0.
314 rows=1001 loops=15)
 |
```

i) This indexing strategy has the same cost as the default index (222), meaning it doesn't usefully affect performance.

4) Advanced query 4

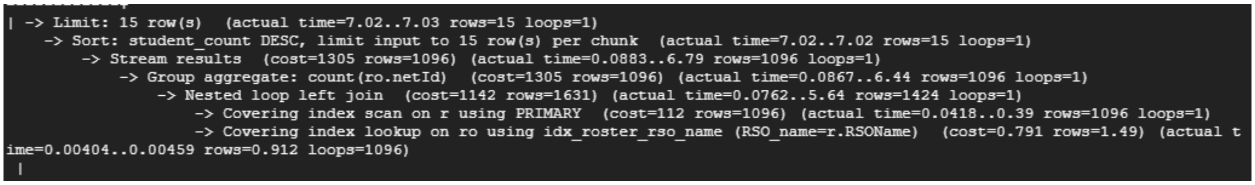
a) CREATE INDEX idx_rso_tagged on RSOs(taggedPref);



```
| -> Limit: 15 row(s)  (actual time=9.13..9.13 rows=15 loops=1)
    -> Sort: student_count DESC, limit input to 15 row(s) per chunk  (actual time=9.13..9.13 rows=15 loops=1)
        -> Stream results  (cost=1305 rows=1096) (actual time=0.23..8.92 rows=1096 loops=1)
            -> Group aggregate: count(ro.netId)  (cost=1305 rows=1096) (actual time=0.226..8.56 rows=1096 loops=1)
                -> Nested loop left join  (cost=1142 rows=1631) (actual time=0.207..7.73 rows=1424 loops=1)
                    -> Covering index scan on r using PRIMARY  (cost=112 rows=1096) (actual time=0.0898..0.519 rows=1096 loops=1)
                    -> Covering index lookup on ro using idx_roster_rso_name (RSO_name=r.RSOName)  (cost=0.791 rows=1.49) (actual t
ime=0.00587..0.00637 rows=0.912 loops=1096)
 |
```

i) This indexing strategy has a higher cost (1305) compared to the default index (727), meaning it has worse performance and is not an ideal indexing strategy.

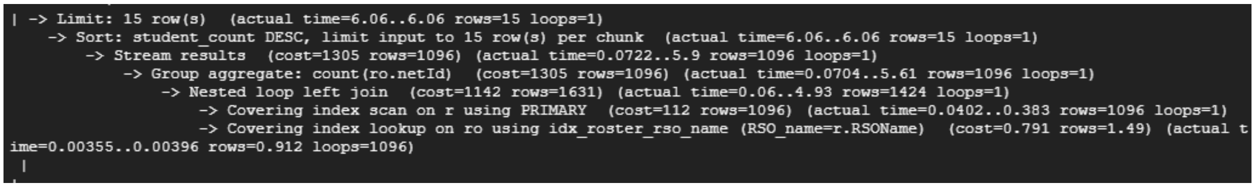
b) CREATE INDEX idx_rso_tagged on RSOs(expTimeComm);



```
| -> Limit: 15 row(s)  (actual time=7.02..7.03 rows=15 loops=1)
    -> Sort: student_count DESC, limit input to 15 row(s) per chunk  (actual time=7.02..7.02 rows=15 loops=1)
        -> Stream results  (cost=1305 rows=1096) (actual time=0.0883..6.79 rows=1096 loops=1)
            -> Group aggregate: count(ro.netId)  (cost=1305 rows=1096) (actual time=0.0867..6.44 rows=1096 loops=1)
                -> Nested loop left join  (cost=1142 rows=1631) (actual time=0.0762..5.64 rows=1424 loops=1)
                    -> Covering index scan on r using PRIMARY  (cost=112 rows=1096) (actual time=0.0418..0.39 rows=1096 loops=1)
                    -> Covering index lookup on ro using idx_roster_rso_name (RSO_name=r.RSOName)  (cost=0.791 rows=1.49) (actual t
ime=0.00404..0.00459 rows=0.912 loops=1096)
 |
```

i) This indexing strategy has a higher cost (1305) compared to the default index (727), meaning it has worse performance and is not an ideal indexing strategy.

c) CREATE INDEX idx_rso_tagged on RSOs(department);



```
| -> Limit: 15 row(s)  (actual time=6.06..6.06 rows=15 loops=1)
    -> Sort: student_count DESC, limit input to 15 row(s) per chunk  (actual time=6.06..6.06 rows=15 loops=1)
        -> Stream results  (cost=1305 rows=1096) (actual time=0.0722..5.9 rows=1096 loops=1)
            -> Group aggregate: count(ro.netId)  (cost=1305 rows=1096) (actual time=0.0704..5.61 rows=1096 loops=1)
                -> Nested loop left join  (cost=1142 rows=1631) (actual time=0.06..4.93 rows=1424 loops=1)
                    -> Covering index scan on r using PRIMARY  (cost=112 rows=1096) (actual time=0.0402..0.383 rows=1096 loops=1)
                    -> Covering index lookup on ro using idx_roster_rso_name (RSO_name=r.RSOName)  (cost=0.791 rows=1.49) (actual t
ime=0.00355..0.00396 rows=0.912 loops=1096)
 |
```

i) This indexing strategy has a higher cost (1305) compared to the default index (727), meaning it has worse performance and is not an ideal indexing strategy.

Final Index Report

1. The indexing strategies we implemented did not improve the performance of this specific query. It seems that our default index is the most optimal. The lack of improvement may be due to the structure of the subquery which includes the NOT IN combined with a UNION clause. The optimizer may prefer to resolve with sequential scans or nested loops instead of our current solution. The indexes we tested introduced additional overhead which increased scan cost and join cost leading to a lesser solution. Leaving the query unindexed would be the best performance for this case.
2. Indexing strategies for our second advanced query also did not lead to performance gains. This is likely due to what the query was designed to do and the optimizer's preference for sequential scans. This query was designed to find a single answer and it may not be improved through indexing. We have decided to leave the query with the default indexing as it seems any customizations either made no difference or negatively impacted our cost.
3. Again for our third advanced query, none of the indexing strategies tested provided improvements to performance. Utilizing a NOT IN, UNION, and filtering across three columns is inherent to the structure of the query and likely made the indexes ineffective. Because of this, no indexing changes for this query perform the best compared to the default indexing.

4. Lastly, all of our indexes for the fourth query had a higher cost than the original cost. The indexes on RSOs(expTimeComm), RSOs(department), or RSOs(taggedPref) for the current query structures, did not contribute to performance and increased total query cost. Instead, focus indexing efforts on attributes directly involved in filtering, joining, or grouping operations.

Resubmission of Stage 2:

Updated stage 2 has been uploaded under the name Stage 2 Resubmit. The required sheet detailing our revisions is under the name stage2_revisions as directed.