Connecting to GCP

gcloud sql connect sp24-cs411-team032 --user=root;
Password: team032-dbsystems
SHOW DATABASES;
USE unirankdb;
SHOW TABLES;

```
maanas_belambe@cloudshell:~ (sp24-cs411-team032)$ gcloud sql connect sp24-cs411-team032 --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 7296
Server version: 8.0.31-google (Google)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

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

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

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| unirankdb          |
+--------------------+
5 rows in set (0.00 sec)

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

Database changed
mysql> SHOW TABLES;
+---------------------+
| Tables_in_unirankdb |
+---------------------+
| Countries           |
| Favorites           |
| OfferedSubjects     |
| Rankings            |
| Subjects            |
| Universities        |
| Users               |
+---------------------+
7 rows in set (0.01 sec)
```

DDL Commands

```
CREATE TABLE Users (
	Username VARCHAR(50),
	Password VARCHAR(50) NOT NULL,
	PRIMARY KEY (Username)
);

CREATE TABLE Countries (
	Name VARCHAR(50),
	LivingCostScore INT,
	SafetyScore INT,
	RightsScore INT,
	ClimateScore INT,
	PRIMARY KEY (Name)
);

CREATE TABLE Subjects (
	Name VARCHAR(50),
	25_Salary INT,
	50_Salary INT,
	75_Salary INT,
	PRIMARY KEY (Name)
);

CREATE TABLE Universities (
	Name VARCHAR(50),
	Location VARCHAR(50),
	OverallScore DECIMAL(4, 1),
	TeachingScore DECIMAL(4, 1),
	ResearchScore DECIMAL(4, 1),
	EmployerScore DECIMAL(4, 1),
	Population INT,
	PRIMARY KEY (Name),
	FOREIGN KEY (Location) REFERENCES Countries(Name)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);
```

```
CREATE TABLE Favorites (
	Username VARCHAR(50),
	University VARCHAR(50),
	PRIMARY KEY (Username, University),
	FOREIGN KEY (Username) REFERENCES Users(Username)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	FOREIGN KEY (University) REFERENCES Universities(Name)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);

CREATE TABLE OfferedSubjects (
	University VARCHAR(50),
	Subject VARCHAR(50),
	PRIMARY KEY (University, Subject),
	FOREIGN KEY (University) REFERENCES Universities(Name)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	FOREIGN KEY (Subject) REFERENCES Subjects(Name)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);

CREATE TABLE Rankings (
	University VARCHAR(50),
	Year INT,
	Ranking INT,
	PRIMARY KEY (University, Year),
	FOREIGN KEY (University) REFERENCES Universities(Name)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);
```

Inserting Data

```
mysql> SELECT COUNT(*) FROM Universities;
+----------+
| COUNT(*) |
+----------+
|     1346 |
+----------+
1 row in set (0.02 sec)

mysql> SELECT COUNT(*) FROM OfferedSubjects;
+----------+
| COUNT(*) |
+----------+
|     5499 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Rankings;
+----------+
| COUNT(*) |
+----------+
|     8134 |
+----------+
1 row in set (0.04 sec)

mysql>
```

Advanced Queries

Advanced Query 1

Finds the living cost and safety scores for countries with multiple universities that are either above average overall (above 60) or strong in a certain area (teaching/research/employer score above 70):

```
SELECT c.Name, AVG(c.LivingCostScore) AS AvgLivingCostScore, AVG(c.SafetyScore) AS AvgSafetyScore
FROM Countries c
JOIN Universities u ON c.Name = u.Location
WHERE u.OverallScore > 60 OR u.TeachingScore > 70 OR u.ResearchScore > 70 OR u.EmployerScore > 70
GROUP BY c.Name
HAVING COUNT(u.Name) >= 2
ORDER BY AvgLivingCostScore;
```

Top 15 Results:

Name	AvgLivingCostScore	AvgSafetyScore
Australia	23.0000	100.0000
Japan	27.0000	97.0000
France	29.0000	93.0000
Belgium	32.0000	92.0000
United Kingdom	34.0000	95.0000
Mexico	34.0000	52.0000
Denmark	35.0000	95.0000
United States	37.0000	85.0000
Canada	37.0000	97.0000
Netherlands	40.0000	96.0000
Sweden	40.0000	96.0000
South Korea	42.0000	95.0000
Switzerland	44.0000	99.0000
Germany	44.0000	95.0000
India	45.0000	79.0000

15 rows in set (0.01 sec)

Advanced Query 2

Most Commonly Offered Subject in Countries with High University Research Scores:

```
SELECT
   c.Name AS CountryName,
   sub.Subject,
   COUNT(*) AS SubjectFrequency
FROM Countries c
JOIN Universities u ON c.Name = u.Location
JOIN OfferedSubjects os ON u.Name = os.University
JOIN (
   SELECT Subject, COUNT(*) AS TotalOfferings
   FROM OfferedSubjects
   GROUP BY Subject
) AS sub ON os.Subject = sub.Subject
WHERE u.ResearchScore > 80
GROUP BY c.Name, sub.Subject
ORDER BY c.Name, SubjectFrequency DESC;
```

Top 15 Results:

CountryName	Subject	SubjectFrequency
Canada	Psychology	1
Canada	General Engineering	1
Canada	Geography	1
Canada	Architecture	1
Canada	History	1
Canada	Chemical Engineering	1
Canada	Chemistry	1
Canada	Civil Engineering	1
Canada	Sociology	1
Canada	Computer Science	1
Germany	Chemistry	1
Germany	Architecture	1
Germany	Computer Science	1
Germany	Civil Engineering	1
Germany	General Engineering	1

Advanced Query 3

Searches through all of the rankings and finds how much a school has improved from the earliest recorded ranking to the most recent recorded ranking. Sorted by descending order of improvement, so schools with the most ranking improvement first.

```
SELECT U.Name,
      (R_earliest.Ranking - R_latest.Ranking) AS Improvement
FROM Universities U
JOIN (
   SELECT University, MIN(Year) AS EarliestYear, MAX(Year) AS LatestYear
   FROM Rankings
   WHERE Year BETWEEN YEAR(CURDATE()) - 5 AND YEAR(CURDATE())
   GROUP BY University
) AS Y ON U.Name = Y.University
JOIN Rankings R_earliest ON U.Name = R_earliest.University AND Y.EarliestYear =
R_earliest.Year
JOIN Rankings R_latest ON U.Name = R_latest.University AND Y.LatestYear = R_latest.Year
WHERE R_earliest.Ranking > R_latest.Ranking
ORDER BY Improvement DESC;
```

Top 15 Results:

Name	Improvement
North South University	10300
Università degli Studi della Tuscia	10000
Rochester Institute of Technology	10000
Taibah University	10000
Mississippi State University	10000
University of Derby	10000
Mutah University	10000
Telkom University	10000
Kookmin University	10000
Kangwon National University	10000
Universitas Hasanuddin	10000
BRAC University	10000
Chungbuk National University	10000
Universita' degli Studi "G. d'Annunzio" Chieti Pes	10000
Imam Mohammad Ibn Saud Islamic University - IMSIU	10000

Advanced Query 4

Selects the universities that offer more than 5 subjects that have a high earning potential, which is measured by having a 75th percentile salary greater than 80,000.

```
SELECT u.Name AS UniversityName, COUNT(*) AS HighSalarySubjects
FROM Universities u
JOIN OfferedSubjects os ON u.Name = os.University
JOIN Subjects s ON os.Subject = s.Name
WHERE s.75_Salary > 80000
GROUP BY u.Name
HAVING COUNT(*) > 5;
```

Top 15 Results:

UniversityName	HighSalarySubjects
Aalborg University	6
Aalto University	6
Abu Dhabi University	6
Aix-Marseille University	6
Al Ain University	6
Al-Balqa Applied University	6
Aligarh Muslim University	6
American University of Beirut	6
American University of Sharjah	6
Amity University	6
Aristotle University of Thessaloniki	6
Arizona State University	6
Auburn University	6
Bandung Institute of Technology	6
Bangladesh University of Engineering and Technolog	6

Indexing Analysis

Query 1

Before

```
-> Sort: AvgLivingCostScore  (actual time=3.843..3.845 rows=22 loops=1)
    -> Filter: (count(u.`Name`) >= 2)  (actual time=3.791..3.801 rows=22 loops=1)
        -> Table scan on <temporary>  (actual time=3.786..3.793 rows=32 loops=1)
            -> Aggregate using temporary table  (actual time=3.784..3.784 rows=32 loops=1
)
                -> Nested loop inner join  (cost=514.87 rows=1080) (actual time=0.847..3.
487 rows=162 loops=1)
                    -> Filter: (((u.OverallScore > 60.0) or (u.TeachingScore > 70.0) or (
u.ResearchScore > 70.0) or (u.EmployerScore > 70.0)) and (u.Location is not null))  (cost
=136.85 rows=1080) (actual time=0.264..2.163 rows=162 loops=1)
                        -> Table scan on u  (cost=136.85 rows=1346) (actual time=0.247..1
.527 rows=1346 loops=1)
                    -> Single-row index lookup on c using PRIMARY (Name=u.Location)  (cos
t=0.25 rows=1) (actual time=0.008..0.008 rows=1 loops=162)
 |
```

Attempt 1 + Analysis

CREATE INDEX idx_universities_researchscore ON Universities(ResearchScore);

```
-> Sort: AvgLivingCostScore  (actual time=1.588..1.591 rows=22 loops=1)
    -> Filter: (count(u.`Name`) >= 2)  (actual time=1.555..1.562 rows=22 loops=1)
        -> Table scan on <temporary>  (actual time=1.552..1.557 rows=32 loops=1)
            -> Aggregate using temporary table  (actual time=1.551..1.551 rows=32 loops=1)
                -> Nested loop inner join  (cost=514.87 rows=1080) (actual time=0.102..1.366 rows=16
2 loops=1)
                    -> Filter: (((u.OverallScore > 60.0) or (u.TeachingScore > 70.0) or (u.ResearchS
core > 70.0) or (u.EmployerScore > 70.0)) and (u.Location is not null))  (cost=136.85 rows=1080) (ac
tual time=0.082..1.127 rows=162 loops=1)
                        -> Table scan on u  (cost=136.85 rows=1346) (actual time=0.074..0.675 rows=1
346 loops=1)
                    -> Single-row index lookup on c using PRIMARY (Name=u.Location)  (cost=0.25 rows
=1) (actual time=0.001..0.001 rows=1 loops=162)
 |
```

After creating an index on the ResearchScore for Universities, the cost did not change, which might be because the threshold of 60 for ResearchScores is leading to a minimally selective result group. Additionally, there are other “OR” conditions present, which means the index may not be useful unless all conditions are on columns that are indexed. However, the time was significantly better after the index was implemented, which is a good sign.

Attempt 2 + Analysis

CREATE INDEX idx_universities_location ON Universities(Location);

```
-------------------------------------------------------------------------+
| -> Sort: AvgLivingCostScore  (actual time=1.640..1.642 rows=22 loops=1)
    -> Filter: (count(u.`Name`) >= 2)  (actual time=1.603..1.611 rows=22 loops=1)
        -> Table scan on <temporary>  (actual time=1.600..1.605 rows=32 loops=1)
            -> Aggregate using temporary table  (actual time=1.598..1.598 rows=32 loops=1)
                -> Nested loop inner join  (cost=514.87 rows=1080) (actual time=0.103..1.407 rows=162 loops=1)
                    -> Filter: (((u.OverallScore > 60.0) or (u.TeachingScore > 70.0) or (u.ResearchScore > 70.0) or (u.Emplo
yerScore > 70.0)) and (u.Location is not null))  (cost=136.85 rows=1080) (actual time=0.084..1.111 rows=162 loops=1)
                        -> Table scan on u  (cost=136.85 rows=1346) (actual time=0.076..0.614 rows=1346 loops=1)
                    -> Single-row index lookup on c using PRIMARY (Name=u.Location)  (cost=0.25 rows=1) (actual time=0.002..
0.002 rows=1 loops=162)
 |
+-------------------------------------------------------------------------------------------------------------------------
```

Once again, the cost did not change after creating an index on the location of the Universities. The time improvement, however, is similar to that of the first indexing attempt for this query.

Attempt 3 + Analysis

CREATE INDEX idx_universities_scores ON Universities(OverallScore, TeachingScore, ResearchScore, EmployerScore);

```
-------------------------------------------------------------------------+
| -> Sort: AvgLivingCostScore  (actual time=1.579..1.581 rows=22 loops=1)
    -> Filter: (count(u.`Name`) >= 2)  (actual time=1.547..1.555 rows=22 loops=1)
        -> Table scan on <temporary>  (actual time=1.544..1.549 rows=32 loops=1)
            -> Aggregate using temporary table  (actual time=1.543..1.543 rows=32 loops=1)
                -> Nested loop inner join  (cost=514.87 rows=1080) (actual time=0.125..1.373 rows=162 loops=1)
                    -> Filter: (((u.OverallScore > 60.0) or (u.TeachingScore > 70.0) or (u.ResearchScore > 70.0) or (u.Emplo
yerScore > 70.0)) and (u.Location is not null))  (cost=136.85 rows=1080) (actual time=0.108..1.140 rows=162 loops=1)
                        -> Table scan on u  (cost=136.85 rows=1346) (actual time=0.101..0.670 rows=1346 loops=1)
                    -> Single-row index lookup on c using PRIMARY (Name=u.Location)  (cost=0.25 rows=1) (actual time=0.001..
0.001 rows=1 loops=162)
 |
+-------------------------------------------------------------------------------------------------------------------------
```

After the first attempt to index, we thought that including all 4 of the attributes in the WHERE clause would improve the performance of the query. However, the cost is once again the same, while the time has improved at a similar level to that of the first two indexing attempts. Overall, this index seems to provide the best performance improvement, albeit a small improvement.

Query 2

Before

```
-------------------------------------------------------------------------+
| -> Sort: c.`Name`, SubjectFrequency DESC  (actual time=18.069..18.073 rows=57 loops=1)
    -> Table scan on <temporary>  (actual time=16.470..16.486 rows=57 loops=1)
        -> Aggregate using temporary table  (actual time=16.467..16.467 rows=57 loops=1)
            -> Nested loop inner join  (cost=1015577.23 rows=10078659) (actual time=2.951..16.148 ro
ws=244 loops=1)
                -> Nested loop inner join  (cost=3121.79 rows=1833) (actual time=0.171..13.071 rows=
244 loops=1)
                    -> Nested loop inner join  (cost=2480.30 rows=1833) (actual time=0.165..12.801 r
ows=244 loops=1)
                        -> Covering index scan on os using idx_offeredsubjects_subject  (cost=555.65
 rows=5499) (actual time=0.046..1.522 rows=5499 loops=1)
                        -> Filter: ((u.ResearchScore > 80.0) and (u.Location is not null))  (cost=0.
25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=5499)
                            -> Single-row index lookup on u using PRIMARY (Name=os.University)  (cos
t=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=5499)
                    -> Single-row covering index lookup on c using PRIMARY (Name=u.Location)  (cost=
0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=244)
                -> Index lookup on sub using <auto_key0> (Subject=os.`Subject`)  (actual time=0.012.
.0.012 rows=1 loops=244)
                    -> Materialize  (cost=1655.45..1655.45 rows=5499) (actual time=2.772..2.772 rows
=10 loops=1)
                        -> Group aggregate: count(0)  (cost=1105.55 rows=5499) (actual time=0.231..2
.753 rows=10 loops=1)
                            -> Covering index scan on OfferedSubjects using idx_offeredsubjects_subj
ect  (cost=555.65 rows=5499) (actual time=0.027..1.422 rows=5499 loops=1)
```

Attempt 1 + Analysis

CREATE INDEX idx_offeredsubjects_subject ON OfferedSubjects(Subject);

```
---------------+
> Sort: c.`Name`, SubjectFrequency DESC  (actual time=8.329..8.333 rows=57 loops=1)
-> Table scan on <temporary>  (actual time=8.254..8.266 rows=57 loops=1)
    -> Aggregate using temporary table  (actual time=8.250..8.250 rows=57 loops=1)
        -> Nested loop inner join  (cost=194982.00 rows=1939826) (actual time=3.682..7.939 rows=244 loops=1)
            -> Nested loop inner join  (cost=116.10 rows=353) (actual time=0.737..4.728 rows=244 loops=1)
                -> Nested loop inner join  (cost=62.35 rows=46) (actual time=0.713..4.278 rows=28 loops=1)
                    -> Covering index scan on c using PRIMARY  (cost=14.05 rows=138) (actual time=0.073..0.128 rows=138 loops=1)
                    -> Filter: (u.ResearchScore > 80.0)  (cost=0.25 rows=0.3) (actual time=0.025..0.030 rows=0 loops=138)
                        -> Index lookup on u using Universities_ibfk_1 (Location=c.`Name`)  (cost=0.25 rows=1) (actual time=0.013
                -> Covering index lookup on os using PRIMARY (University=u.`Name`)  (cost=0.42 rows=8) (actual time=0.013..0.015 r
            -> Index lookup on sub using <auto_key0> (Subject=os.`Subject`)  (actual time=0.013..0.013 rows=1 loops=244)
                -> Materialize  (cost=1673.95..1673.95 rows=5499) (actual time=2.936..2.936 rows=10 loops=1)
                    -> Group aggregate: count(0)  (cost=1124.05 rows=5499) (actual time=0.267..2.910 rows=10 loops=1)
                        -> Covering index scan on OfferedSubjects using Subject  (cost=574.15 rows=5499) (actual time=0.043..1.475
```

The reason the cost went down significantly could be because the index placed on Subject can scan for results much faster if going through the data by only indexing that column. Also, the JOIN allows for matches to be found between OfferedSubjects and Subject results.

Attempt 2 + Analysis

CREATE INDEX idx_universities_researchscore ON Universities(ResearchScore);

```
| -> Sort: c.`Name`, SubjectFrequency DESC  (actual time=3.948..3.952 rows=57 loops=1)
    -> Table scan on <temporary>  (actual time=3.899..3.908 rows=57 loops=1)
        -> Aggregate using temporary table  (actual time=3.898..3.898 rows=57 loops=1)
            -> Nested loop inner join  (cost=118681.27 rows=1180881) (actual time=2.869..3.614 rows=
244 loops=1)
                -> Nested loop inner join  (cost=55.38 rows=215) (actual time=0.114..0.619 rows=244
loops=1)
                    -> Nested loop inner join  (cost=22.66 rows=28) (actual time=0.090..0.235 rows=2
8 loops=1)
                        -> Filter: (u.Location is not null)  (cost=12.86 rows=28) (actual time=0.077
..0.185 rows=28 loops=1)
                            -> Index range scan on u using idx_universities_researchscore_name over
(80.0 < ResearchScore), with index condition: (u.ResearchScore > 80.0)  (cost=12.86 rows=28) (actual
 time=0.041..0.145 rows=28 loops=1)
                        -> Single-row covering index lookup on c using PRIMARY (Name=u.Location)  (c
ost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=28)
                    -> Covering index lookup on os using PRIMARY (University=u.`Name`)  (cost=0.43 r
ows=8) (actual time=0.011..0.013 rows=9 loops=28)
                -> Index lookup on sub using <auto_key0> (Subject=os.`Subject`)  (actual time=0.012.
.0.012 rows=1 loops=244)
                    -> Materialize  (cost=1655.45..1655.45 rows=5499) (actual time=2.749..2.749 rows
=10 loops=1)
                        -> Group aggregate: count(0)  (cost=1105.55 rows=5499) (actual time=0.241..2
.732 rows=10 loops=1)
                            -> Covering index scan on OfferedSubjects using idx_offeredsubjects_subj
ect  (cost=555.65 rows=5499) (actual time=0.032..1.394 rows=5499 loops=1)
```

The cost significantly went down with this index as well, it could be because the index starts with ResearchScore which is the first column and helps select criteria matches much faster.

Attempt 3 + Analysis

CREATE INDEX idx_universities_location ON Universities(Location);

```
+-------------------------------------------------------------------------+
| -> Sort: c.`Name`, SubjectFrequency DESC  (actual time=14.943..14.968 rows=57 loops=1)
    -> Table scan on <temporary>  (actual time=14.889..14.900 rows=57 loops=1)
        -> Aggregate using temporary table  (actual time=14.885..14.885 rows=57 loops=1)
            -> Nested loop inner join  (cost=1015577.23 rows=10078659) (actual time=2.904..14.606 ro
ws=244 loops=1)
                -> Nested loop inner join  (cost=3121.79 rows=1833) (actual time=0.147..11.589 rows=
244 loops=1)
                    -> Nested loop inner join  (cost=2480.30 rows=1833) (actual time=0.141..11.330 r
ows=244 loops=1)
                        -> Covering index scan on os using idx_offeredsubjects_subject  (cost=555.65
 rows=5499) (actual time=0.042..1.398 rows=5499 loops=1)
                        -> Filter: ((u.ResearchScore > 80.0) and (u.Location is not null))  (cost=0.
25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=5499)
                            -> Single-row index lookup on u using PRIMARY (Name=os.University)  (cos
t=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=5499)
                    -> Single-row covering index lookup on c using PRIMARY (Name=u.Location)  (cost=
0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=244)
                -> Index lookup on sub using <auto_key0> (Subject=os.`Subject`)  (actual time=0.012.
.0.012 rows=1 loops=244)
                    -> Materialize  (cost=1655.45..1655.45 rows=5499) (actual time=2.750..2.750 rows
=10 loops=1)
                        -> Group aggregate: count(0)  (cost=1105.55 rows=5499) (actual time=0.235..2
.735 rows=10 loops=1)
                            -> Covering index scan on OfferedSubjects using idx_offeredsubjects_subj
ect  (cost=555.65 rows=5499) (actual time=0.027..1.405 rows=5499 loops=1)
```

The cost did not change with this index, which could be because the location column has a foreign key constraint being maintained so adding this index won't be more efficient than the one currently there.

Query 3

Before

```
-> Sort: Improvement DESC  (actual time=32.039..32.067 rows=486 loops=1)
    -> Stream results  (cost=1578.85 rows=452) (actual time=23.872..31.823 rows=486 loops=1)
        -> Nested loop inner join  (cost=1578.85 rows=452) (actual time=23.863..31.638 rows=486 loops=1)
            -> Nested loop inner join  (cost=1104.25 rows=1356) (actual time=23.827..28.963 rows=1266 loop
s=1)
                -> Nested loop inner join  (cost=629.65 rows=1356) (actual time=23.811..26.471 rows=1266 l
oops=1)
                    -> Filter: ((Y.EarliestYear is not null) and (Y.LatestYear is not null))  (cost=686.31
..155.05 rows=1356) (actual time=23.782..24.202 rows=1266 loops=1)
                        -> Table scan on Y  (cost=686.71..706.15 rows=1356) (actual time=23.774..24.053 ro
ws=1266 loops=1)
                            -> Materialize  (cost=686.70..686.70 rows=1356) (actual time=23.769..23.769 ro
ws=1266 loops=1)
                                -> Filter: (Rankings.`Year` between <cache>((year(curdate()) - 5)) and <ca
che>(year(curdate())))  (cost=551.10 rows=1356) (actual time=0.100..23.115 rows=1266 loops=1)
                                    -> Covering index skip scan for grouping on Rankings using PRIMARY ove
r (2019 <= Year <= 2024)  (cost=551.10 rows=1356) (actual time=0.097..22.952 rows=1266 loops=1)
                    -> Single-row covering index lookup on U using PRIMARY (Name=Y.University)  (cost=0.25
 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
                -> Single-row index lookup on R_earliest using PRIMARY (University=Y.University, Year=Y.Ea
rliestYear)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
            -> Filter: (R_earliest.Ranking > R_latest.Ranking)  (cost=0.25 rows=0.3) (actual time=0.002..0
.002 rows=0 loops=1266)
                -> Single-row index lookup on R_latest using PRIMARY (University=Y.University, Year=Y.Late
stYear)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
```

Attempt 1 + Analysis

CREATE INDEX idx_rankings_university_year ON Rankings(University, Year);

```
-> Sort: Improvement DESC  (actual time=32.039..32.067 rows=486 loops=1)
    -> Stream results  (cost=1578.85 rows=452) (actual time=23.872..31.823 rows=486 loops=1)
        -> Nested loop inner join  (cost=1578.85 rows=452) (actual time=23.863..31.638 rows=486 loops=1)
            -> Nested loop inner join  (cost=1104.25 rows=1356) (actual time=23.827..28.963 rows=1266 loop
s=1)
                -> Nested loop inner join  (cost=629.65 rows=1356) (actual time=23.811..26.471 rows=1266 l
oops=1)
                    -> Filter: ((Y.EarliestYear is not null) and (Y.LatestYear is not null))  (cost=686.31
..155.05 rows=1356) (actual time=23.782..24.202 rows=1266 loops=1)
                        -> Table scan on Y  (cost=686.71..706.15 rows=1356) (actual time=23.774..24.053 ro
ws=1266 loops=1)
                            -> Materialize  (cost=686.70..686.70 rows=1356) (actual time=23.769..23.769 ro
ws=1266 loops=1)
                                -> Filter: (Rankings.`Year` between <cache>((year(curdate()) - 5)) and <ca
che>(year(curdate())))  (cost=551.10 rows=1356) (actual time=0.100..23.115 rows=1266 loops=1)
                                    -> Covering index skip scan for grouping on Rankings using PRIMARY ove
r (2019 <= Year <= 2024)  (cost=551.10 rows=1356) (actual time=0.097..22.952 rows=1266 loops=1)
                    -> Single-row covering index lookup on U using PRIMARY (Name=Y.University)  (cost=0.25
 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
                -> Single-row index lookup on R_earliest using PRIMARY (University=Y.University, Year=Y.Ea
rliestYear)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
            -> Filter: (R_earliest.Ranking > R_latest.Ranking)  (cost=0.25 rows=0.3) (actual time=0.002..0
.002 rows=0 loops=1266)
                -> Single-row index lookup on R_latest using PRIMARY (University=Y.University, Year=Y.Late
stYear)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
```

This index may not have had an impact on the performance because it may think the primary key is already efficient in its role, so the index isn't being used, or it could be because the distribution of the data doesn't allow for enhancement with the index.

Attempt 2 + Analysis

CREATE INDEX idx_rankings_year_max ON Rankings(Year DESC);

```
| -> Sort: Improvement DESC  (actual time=39.314..39.357 rows=486 loops=1)
    -> Stream results  (cost=3552.78 rows=1018) (actual time=20.405..38.950 rows=486 loops=1)
        -> Nested loop inner join  (cost=3552.78 rows=1018) (actual time=20.396..38.630 rows=486 loops=1)
            -> Nested loop inner join  (cost=2483.88 rows=3054) (actual time=20.357..29.039 rows=1266 loop
s=1)
                -> Nested loop inner join  (cost=1414.98 rows=3054) (actual time=20.335..24.857 rows=1266
loops=1)
                    -> Filter: ((Y.EarliestYear is not null) and (Y.LatestYear is not null))  (cost=0.11..
346.07 rows=3054) (actual time=20.292..21.022 rows=1266 loops=1)
                        -> Table scan on Y  (cost=2.50..2.50 rows=0) (actual time=20.285..20.769 rows=1266
 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=20.283..20.283 rows=1266
 loops=1)
                                -> Table scan on <temporary>  (actual time=19.360..19.694 rows=1266 loops=
1)
                                    -> Aggregate using temporary table  (actual time=19.354..19.354 rows=1
266 loops=1)
                                        -> Filter: (Rankings.`Year` between <cache>((year(curdate()) - 5))
 and <cache>(year(curdate())))  (cost=685.08 rows=3054) (actual time=0.039..4.991 rows=6492 loops=1)
                                            -> Covering index range scan on Rankings using idx_rankings_ye
ar_max over (2024 <= Year <= 2019)  (cost=685.08 rows=3054) (actual time=0.033..4.084 rows=6492 loops=1)
                    -> Single-row covering index lookup on U using PRIMARY (Name=Y.University)  (cost=0.25
 rows=1) (actual time=0.003..0.003 rows=1 loops=1266)
                -> Single-row index lookup on R_earliest using PRIMARY (University=Y.University, Year=Y.Ea
rliestYear)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1266)
            -> Filter: (R_earliest.Ranking > R_latest.Ranking)  (cost=0.25 rows=0.3) (actual time=0.007..0
.007 rows=0 loops=1266)
                -> Single-row index lookup on R_latest using PRIMARY (University=Y.University, Year=Y.Late
stYear)  (cost=0.25 rows=1) (actual time=0.007..0.007 rows=1 loops=1266)
 |
```

The performance may have decreased because the index doesn't cover all the columns that query is using, or because it prefers to use the primary key index of the group rather than a one column index like this one.

Attempt 3 + Analysis

CREATE INDEX idx_rankings_year ON Rankings(Year);

```
| -> Sort: Improvement DESC  (actual time=19.447..19.476 rows=486 loops=1)
    -> Stream results  (cost=3552.78 rows=1018) (actual time=10.578..19.219 rows=486 loops=1)
        -> Nested loop inner join  (cost=3552.78 rows=1018) (actual time=10.570..19.027 rows=486 loops=1)
            -> Nested loop inner join  (cost=2483.88 rows=3054) (actual time=10.546..16.088 rows=1266 loop
s=1)
                -> Nested loop inner join  (cost=1414.98 rows=3054) (actual time=10.447..13.336 rows=1266
loops=1)
                    -> Filter: ((Y.EarliestYear is not null) and (Y.LatestYear is not null))  (cost=0.11..
346.07 rows=3054) (actual time=10.407..10.830 rows=1266 loops=1)
                        -> Table scan on Y  (cost=2.50..2.50 rows=0) (actual time=10.402..10.678 rows=1266
 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=10.400..10.400 rows=1266
 loops=1)
                                -> Table scan on <temporary>  (actual time=9.844..10.038 rows=1266 loops=1
)
                                    -> Aggregate using temporary table  (actual time=9.841..9.841 rows=126
6 loops=1)
                                        -> Filter: (Rankings.`Year` between <cache>((year(curdate()) - 5))
 and <cache>(year(curdate())))  (cost=685.08 rows=3054) (actual time=0.030..4.551 rows=6492 loops=1)
                                            -> Covering index range scan on Rankings using idx_rankings_ye
ar over (2019 <= Year <= 2024)  (cost=685.08 rows=3054) (actual time=0.026..3.682 rows=6492 loops=1)
                    -> Single-row covering index lookup on U using PRIMARY (Name=Y.University)  (cost=0.25
 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
                -> Single-row index lookup on R_earliest using PRIMARY (University=Y.University, Year=Y.Ea
rliestYear)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
            -> Filter: (R_earliest.Ranking > R_latest.Ranking)  (cost=0.25 rows=0.3) (actual time=0.002..0
.002 rows=0 loops=1266)
                -> Single-row index lookup on R_latest using PRIMARY (University=Y.University, Year=Y.Late
stYear)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=1266)
 |
```

The cost may have gone up because the year column may have low cardinality, or it may be that it doesn't cause enough restriction in the results, so it's not as selective, which is costly if using the index.

Query 4

Before

```
-----------------------------------------------------------------------------------------------------
---------------------------------+
| -> Filter: (count(0) > 5)  (actual time=14.457..14.617 rows=292 loops=1)
    -> Table scan on <temporary>  (actual time=14.453..14.556 rows=715 loops=1)
        -> Aggregate using temporary table  (actual time=14.449..14.449 rows=715 loops=1)
            -> Nested loop inner join  (cost=3140.29 rows=1833) (actual time=0.327..11.999 rows=3338
 loops=1)
                -> Nested loop inner join  (cost=2498.80 rows=1833) (actual time=0.316..6.437 rows=3
338 loops=1)
                    -> Covering index scan on os using Subject  (cost=574.15 rows=5499) (actual time
=0.151..2.772 rows=5499 loops=1)
                    -> Filter: (s.75_Salary > 80000)  (cost=0.25 rows=0.3) (actual time=0.000..0.001
 rows=1 loops=5499)
                        -> Single-row index lookup on s using PRIMARY (Name=os.`Subject`)  (cost=0.2
5 rows=1) (actual time=0.000..0.000 rows=1 loops=5499)
                -> Single-row covering index lookup on u using PRIMARY (Name=os.University)  (cost=0
.25 rows=1) (actual time=0.001..0.001 rows=1 loops=3338)
 |
+-----------------------------------------------------------------------------------------------------
```

Attempt 1 + Analysis

CREATE INDEX idx_subjects_75_salary ON Subjects(75_Salary);

```
---------------------------------+
| -> Filter: (count(0) > 5)  (actual time=21.926..22.164 rows=292 loops=1)
    -> Table scan on <temporary>  (actual time=21.923..22.081 rows=715 loops=1)
        -> Aggregate using temporary table  (actual time=21.918..21.918 rows=715 loops=1)
            -> Nested loop inner join  (cost=3328.39 rows=2370) (actual time=0.085..17.283 rows=3338
 loops=1)
                -> Nested loop inner join  (cost=2498.80 rows=2370) (actual time=0.075..8.095 rows=3
338 loops=1)
                    -> Covering index scan on os using Subject  (cost=574.15 rows=5499) (actual time
=0.054..2.769 rows=5499 loops=1)
                    -> Filter: (s.75_Salary > 80000)  (cost=0.25 rows=0.4) (actual time=0.001..0.001
 rows=1 loops=5499)
                        -> Single-row index lookup on s using PRIMARY (Name=os.`Subject`)  (cost=0.2
5 rows=1) (actual time=0.000..0.000 rows=1 loops=5499)
                -> Single-row covering index lookup on u using PRIMARY (Name=os.University)  (cost=0
.25 rows=1) (actual time=0.002..0.002 rows=1 loops=3338)
 |
```

The cost went up with this index, which could be because a larger number of rows are satisfying the condition in the WHERE clause, meaning the index is not seen as efficient, or if the 80,000 check is met by several rows, the selectivity goes down which means the index costs more.

Attempt 2 + Analysis

CREATE INDEX idx_universities_researchscore ON Universities(ResearchScore);

```
-> Filter: (count(0) > 5)  (actual time=12.667..12.946 rows=292 loops=1)
    -> Table scan on <temporary>  (actual time=12.664..12.860 rows=715 loops=1)
        -> Aggregate using temporary table  (actual time=12.660..12.660 rows=715 loops=1)
            -> Nested loop inner join  (cost=3123.35 rows=1833) (actual time=0.081..10.285 rows=3338 loops=
1)
                -> Nested loop inner join  (cost=2481.87 rows=1833) (actual time=0.073..5.041 rows=3338 loo
ps=1)
                    -> Covering index scan on os using idx_offeredsubjects_subject  (cost=557.22 rows=5499)
 (actual time=0.051..1.554 rows=5499 loops=1)
                    -> Filter: (s.75_Salary > 80000)  (cost=0.25 rows=0.3) (actual time=0.000..0.000 rows=1
 loops=5499)
                        -> Single-row index lookup on s using PRIMARY (Name=os.`Subject`)  (cost=0.25 rows=
1) (actual time=0.000..0.000 rows=1 loops=5499)
                -> Single-row index lookup on u using PRIMARY (Name=os.University)  (cost=0.25 row
s=1) (actual time=0.001..0.001 rows=1 loops=3338)
 |
```

The cost decreased with this index which could be because without needing to parse the whole table we are able to group the universities that meet the ResearchScore threshold faster which results in less rows to go through. The time is slightly better in the query after the indexing.

Attempt 3 + Analysis

CREATE INDEX idx_universities_overallscore ON Universities(OverallScore);

```
-> Filter: (count(0) > 5)  (actual time=14.852..15.017 rows=292 loops=1)
    -> Table scan on <temporary>  (actual time=14.847..14.964 rows=715 loops=1)
        -> Aggregate using temporary table  (actual time=14.842..14.842 rows=715 loops=1)
            -> Nested loop inner join  (cost=3121.79 rows=1833) (actual time=0.108..11.878 rows=3338
 loops=1)
                -> Nested loop inner join  (cost=2480.30 rows=1833) (actual time=0.098..5.746 rows=3
338 loops=1)
                    -> Covering index scan on os using idx_offeredsubjects_subject  (cost=555.65 row
s=5499) (actual time=0.069..1.988 rows=5499 loops=1)
                    -> Filter: (s.75_Salary > 80000)  (cost=0.25 rows=0.3) (actual time=0.000..0.001
 rows=1 loops=5499)
                        -> Single-row index lookup on s using PRIMARY (Name=os.`Subject`)  (cost=0.2
5 rows=1) (actual time=0.000..0.000 rows=1 loops=5499)
                -> Single-row covering index lookup on u using PRIMARY (Name=os.University)  (cost=0
.25 rows=1) (actual time=0.002..0.002 rows=1 loops=3338)
 |
```

This decreased the cost which could be because the OverallScore threshold of 60 is searched much faster due to the index. Without the index, there is a much larger number of schools that need to be searched to see if they meet the criteria. The following JOINS have less rows to go through because there's less universities in the results so far.