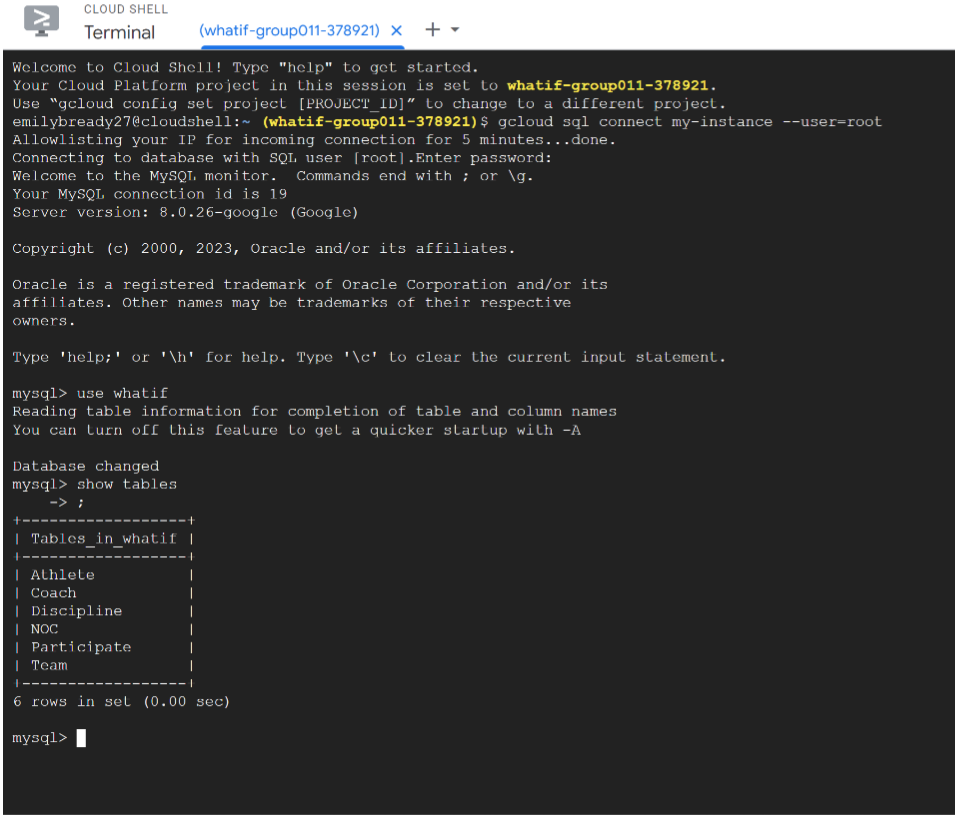
Group 11: HeeHeeHawHaw

Maninder Singh Deol
Hansen Liu
Shloaka Duvvuri
Emily Bready

1. Database implementation

a. Connection to DB with screen shot

i.



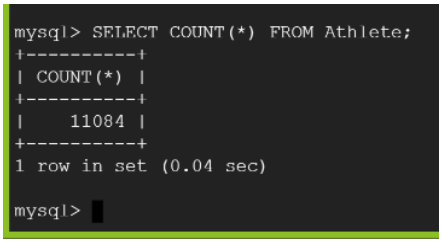
b. DDL commands to create tables

```
CREATE TABLE NOC (
        NOCName VARCHAR(128) PRIMARY KEY,
        ranking INT,
        weightedRanking INT,
        goldMedalCount INT,
        silverMedalCount INT,
        bronzeMedalCount INT,
        totalMedalCount INT
);
CREATE TABLE Discipline (
        discName VARCHAR(128) PRIMARY KEY,
        femalePlayerCount INT,
        malePlayerCount INT,
        totalPlayerCount INT
);
CREATE TABLE Coach (
        coachName VARCHAR(128),
        event VARCHAR(128),
        NOCName VARCHAR(128),
```

```
        discName VARCHAR(128),
        PRIMARY KEY (NOCName, discName, coachName, event),
        FOREIGN KEY (NOCName) REFERENCES NOC(NOCName),
        FOREIGN KEY (discName) REFERENCES Discipline(discName)
);
CREATE TABLE Athlete (
        athleteName VARCHAR(128),
        NOCName VARCHAR(128),
        discName VARCHAR(128),
        PRIMARY KEY (athleteName, NOCName, discName),
        FOREIGN KEY (NOCName) REFERENCES NOC(NOCName),
        FOREIGN KEY (discName) REFERENCES Discipline(discName)
);
CREATE TABLE Participate (
        NOCName VARCHAR(128),
        discName VARCHAR(128),
        PRIMARY KEY (NOCName, discName),
        FOREIGN KEY (NOCName) REFERENCES NOC(NOCName),
        FOREIGN KEY (discName) REFERENCES Discipline(discName)
);
CREATE TABLE Team (
        teamName VARCHAR(128),
        event VARCHAR(128),
        NOCName VARCHAR(128),
        discName VARCHAR(128),
        PRIMARY KEY (NOCName, discName, teamName, event),
        FOREIGN KEY (NOCName) REFERENCES NOC(NOCName) ON
        DELETE CASCADE,
        FOREIGN KEY (discName) REFERENCES Discipline(discName) ON
        DELETE CASCADE
);
```

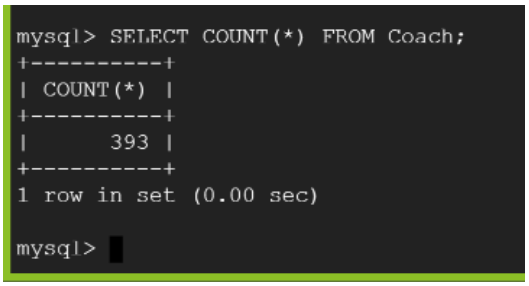
c. Showing a thousand records per table

i.



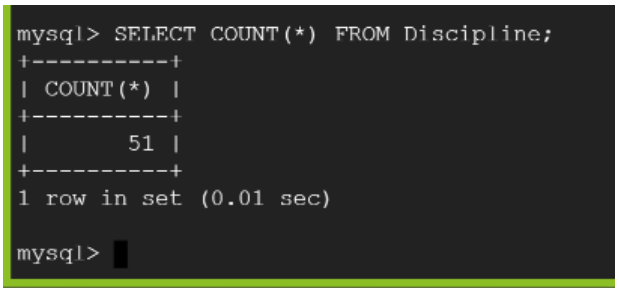
```
mysql> SELECT COUNT(*) FROM Athlete;
+----------+
| COUNT(*) |
+----------+
|    11084 |
+----------+
1 row in set (0.04 sec)

mysql>
```



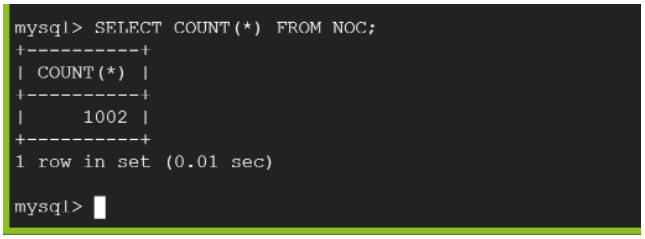
```
mysql> SELECT COUNT(*) FROM Coach;
+----------+
| COUNT(*) |
+----------+
|      393 |
+----------+
1 row in set (0.00 sec)

mysql>
```



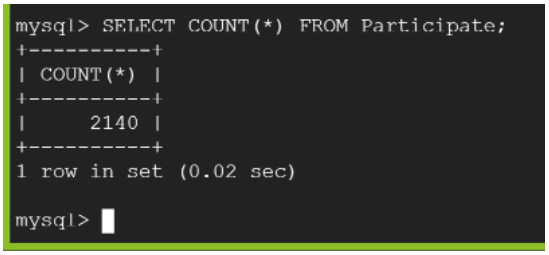
```
mysql> SELECT COUNT(*) FROM Discipline;
+----------+
| COUNT(*) |
+----------+
|       51 |
+----------+
1 row in set (0.01 sec)

mysql>
```



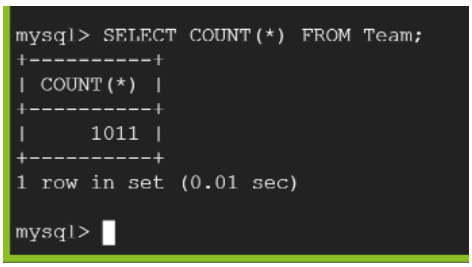
```
mysql> SELECT COUNT(*) FROM NOC;
+----------+
| COUNT(*) |
+----------+
|     1002 |
+----------+
1 row in set (0.01 sec)

mysql>
```



```
mysql> SELECT COUNT(*) FROM Participate;
+----------+
| COUNT(*) |
+----------+
|     2140 |
+----------+
1 row in set (0.02 sec)

mysql>
```



```
mysql> SELECT COUNT(*) FROM Team;
+----------+
| COUNT(*) |
+----------+
|     1011 |
+----------+
1 row in set (0.01 sec)

mysql>
```

2. Advanced Queries

a. **Query 1 (Country and their gold medal count (CGMC)):**

i. SQL command:

```
SELECT n.NOCName, n.goldMedalCount
FROM NOC n NATURAL JOIN Athlete a
WHERE n.goldMedalCount > 0 AND n.goldMedalCount > (
        SELECT AVG(n1.goldMedalCount) AS AverageCount
        FROM NOC n1 NATURAL JOIN Athlete a
        WHERE a.discName = 'Karate'
)
GROUP BY n.NOCName
ORDER BY n.goldMedalCount DESC;
```

ii. Results: (our output only had 10 rows)

NOCName	goldMedalCount
United States of America	39
People's Republic of China	38
Japan	27
Great Britain	22
ROC	20
Australia	17
France	10
Germany	10
Ttaly	10
Netherlands	10

10 rows in set (0.01 sec)

mysql>

b. **Query 2 (Athlete and Coach for B Ball and Football Union (UNION)):**

i. SQL command:

```
(SELECT coachName, c.NOCName, c.discName,
            COUNT(athleteName) AS athleteCount
FROM Athlete a JOIN Coach c ON (a.NOCName = c.NOCName
                                AND a.discName = c.discName)
WHERE a.discName = 'Basketball'
GROUP BY coachName, c.NOCName, c.discName)
UNION
(SELECT coachName, c.NOCName, c.discName,
            COUNT(athleteName) AS athleteCount
FROM Athlete a JOIN Coach c ON (a.NOCName = c.NOCName
```

```
                                                AND a.discName = c.discName)
        WHERE a.discName = 'Football'
        GROUP BY coachName, c.NOCName, c.discName)
        ORDER BY coachName
        LIMIT 15;
```

ii. Results

```
+-----------------------------+-------------------------+------------+--------------+
| coachName                   | NOCName                 | discName   | athleteCount |
+-----------------------------+-------------------------+------------+--------------+
| ABDELMAGID Wael             | Egypt                   | Football   |           20 |
| ABE Katsuhiko               | Japan                   | Basketball |           23 |
| ADAMA Cherif                | Ivory Coast             | Football   |           20 |
| ALLER CARBALLO Manuel Angel | Spain                   | Basketball |           24 |
| ALSHEHRI Saad               | Saudi Arabia            | Football   |           21 |
| ALY Kamal                   | Egypt                   | Football   |           20 |
| AMAYA GAITAN Fabian         | Puerto Rico             | Basketball |           12 |
| AMO AGUADO Pablo            | Spain                   | Football   |           19 |
| ANDONOVSKI Vlatko           | United States of America | Football   |           20 |
| ARNOLD Graham               | Australia               | Football   |           43 |
| BACTU Horatiu               | Romania                 | Football   |           20 |
| BATISTA Fernando            | Argentina               | Football   |           19 |
| BATISTA SANTIAGO Gerardo    | Puerto Rico             | Basketball |           12 |
| BENCTC Filip                | Serbia                  | Basketball |           12 |
| BOLTON Alice                | Nigeria                 | Basketball |           23 |
+-----------------------------+-------------------------+------------+--------------+
15 rows in set (0.02 sec)
```

3. Index Analysis:

i. Pertaining to CGMC query

Before

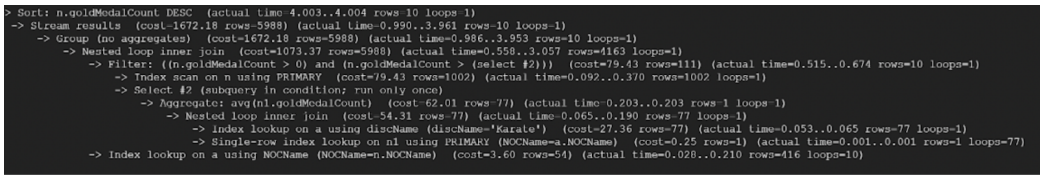
```
| -> Sort: n.goldMedalCount DESC  (actual time=4.110..4.110 rows=10 loops=1)
    -> Stream results  (cost=1672.18 rows=5988) (actual time=0.997..4.080 rows=10 loops=1)
        -> Group (no aggregates)  (cost=1672.18 rows=5988) (actual time=0.994..4.071 rows=10 loops=1)
            -> Nested loop inner join  (cost=1073.37 rows=5988) (actual time=0.538..3.126 rows=4163 loops=1)
                -> Filter: ((n.goldMedalCount > 0) and (n.goldMedalCount > (select #2)))  (cost=79.43 rows=111) (actual time=0.494..0.676 rows=10 loops=1)
                    -> Index scan on n using PRIMARY  (cost=79.43 rows=1002) (actual time=0.040..0.339 rows=1002 loops=1)
                    -> Select #2 (subquery in condition; run only once)
                        -> Aggregate: avg(n1.goldMedalCount)  (cost=62.01 rows=77) (actual time=0.193..0.193 rows=1 loops=1)
                            -> Nested loop inner join  (cost=54.31 rows=77) (actual time=0.059..0.180 rows=77 loops=1)
                                -> Index lookup on a using discName (discName='Karate')  (cost=27.36 rows=77) (actual time=0.039..0.052 rows=77 loops=1)
                                -> Single-row index lookup on n1 using PRIMARY (NOCName=a.NOCName)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=77)
                -> Index lookup on a using NOCName (NOCName=n.NOCName)  (cost=3.60 rows=54) (actual time=0.026..0.217 rows=416 loops=10)
```

After

```
| -> Sort: n.goldMedalCount DESC  (actual time=3.886..3.887 rows=10 loops=1)
    -> Stream results  (cost=1672.18 rows=5988) (actual time=0.853..3.865 rows=10 loops=1)
        -> Group (no aggregates)  (cost=1672.18 rows=5988) (actual time=0.850..3.857 rows=10 loops=1)
            -> Nested loop inner join  (cost=1073.37 rows=5988) (actual time=0.431..2.914 rows=4163 loops=1)
                -> Filter: ((n.goldMedalCount > 0) and (n.goldMedalCount > (select #2)))  (cost=79.43 rows=111) (actual time=0.370..0.525 rows=10 loops=1)
                    -> Index scan on n using PRIMARY  (cost=79.43 rows=1002) (actual time=0.114..0.427 rows=1002 loops=1)
                    -> Select #2 (subquery in condition; run only once)
                        -> Index scan on TempTable using Temp_Index  (cost=0.35 rows=1) (actual time=0.010..0.011 rows=1 loops=1)
                -> Index lookup on a using NOCName (NOCName=n.NOCName)  (cost=3.60 rows=54) (actual time=0.030..0.213 rows=416 loops=10)
```

1. **Indexes design/philosophy**
   i. Design 1:
      With index for Athlete and NOC

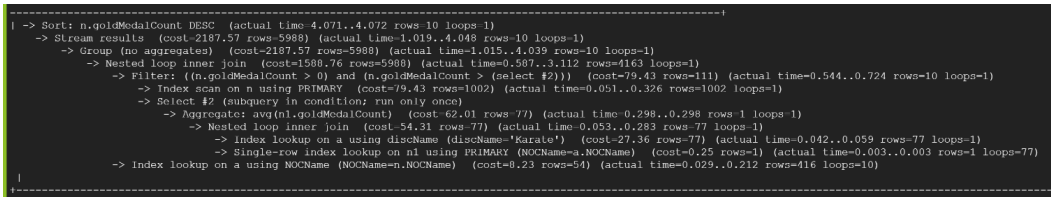
- ☐ CREATE INDEX Athlete_Index ON Athlete (athleteName, NOCName, discName);
- ☐ CREATE INDEX NOC_Index ON NOC (NOCName);



ii. Design 2:

With index for Athlete only

- ☐ CREATE INDEX Athlete_Index ON Athlete (athleteName, NOCName, discName);



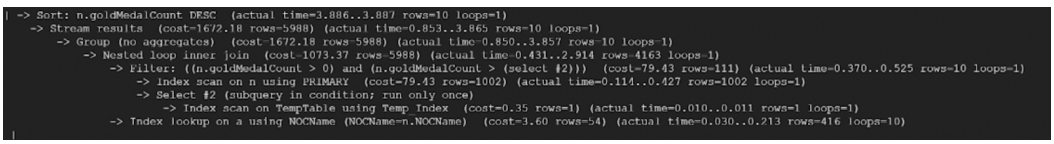
iii. Design 3:

Updated SQL command:

```
SELECT n.NOCName, n.goldMedalCount
        FROM NOC n NATURAL JOIN Athlete a
        WHERE n.goldMedalCount > 0 AND n.goldMedalCount >
        (       SELECT AverageCount
                FROM TempTable      )
GROUP BY n.NOCName
ORDER BY n.goldMedalCount DESC;
```

With index for Athlete, NOC and TempTable (inner query)

- ☐ CREATE TEMPORARY TABLE TempTable
  SELECT AVG(n1.goldMedalCount) AS AverageCount
  FROM NOC n1 NATURAL JOIN Athlete a
  WHERE a.discName = 'Karate';
- ☐ CREATE INDEX TempTable_Index ON TempTable (AverageCount);



Every time we added our new design indexing the actual time did decrease after each one. It did not affect the loops at all. One interesting thing to note is that out design 2 made the cost a lot worse and design 3 fixed that issues having it once again match with before but with an improved actual time.

ii. Pertaining to UNION query

Before

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.138..0.140 rows=15 loops=1)
    -> Sort: coachName, limit input to 15 row(s) per chunk  (cost=2.50 rows=0) (actual time=0.137..0.139 rows=15 loops=1)
        -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.014 rows=123 loops=1)
            -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=25.597..25.600 rows=123 loops=1)
                -> Table scan on <temporary>  (actual time=0.002..0.016 rows=64 loops=1)
                    -> Aggregate using temporary table  (actual time=14.503..14.520 rows=64 loops=1)
                        -> Nested loop inner join  (cost=724.38 rows=101) (actual time=0.659..13.146 rows=1237 loops=1)
                            -> Index lookup on c using discName (discName='Basketball')  (cost=12.21 rows=74) (actual time=0.407..0.453 rows=74 loops=1)
                            -> Filter: (a.discName = 'Basketball')  (cost=4.25 rows=1) (actual time=0.028..0.170 rows=17 loops=74)
                                -> Index lookup on a using NOCName (NOCName=c.NOCName)  (cost=4.25 rows=54) (actual time=0.023..0.138 rows=276 loops=74)
                -> Table scan on <temporary>  (actual time=0.003..0.015 rows=59 loops=1)
                    -> Aggregate using temporary table  (actual time=10.811..10.826 rows=59 loops=1)
                        -> Nested loop inner join  (cost=577.58 rows=162) (actual time=0.155..9.140 rows=1439 loops=1)
                            -> Index lookup on c using discName (discName='Football')  (cost=9.78 rows=59) (actual time=0.109..0.144 rows=59 loops=1)
                            -> Filter: (a.discName = 'Football')  (cost=4.25 rows=3) (actual time=0.031..0.151 rows=24 loops=59)
                                -> Index lookup on a using NOCName (NOCName=c.NOCName)  (cost=4.25 rows=54) (actual time=0.023..0.123 rows=244 loops=59)
 |
```

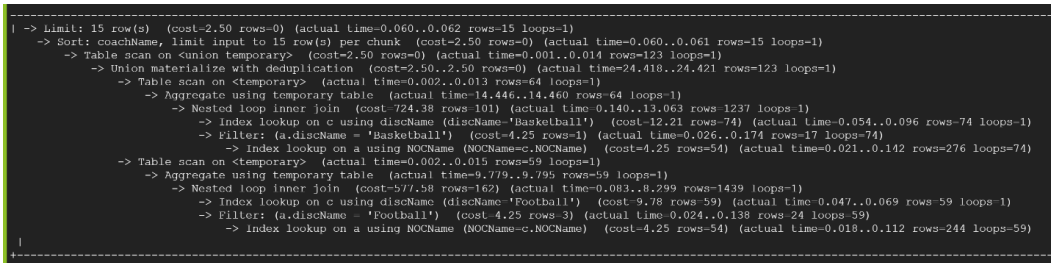
After

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.077..0.079 rows=15 loops=1)
    -> Sort: coachName, limit input to 15 row(s) per chunk  (cost=2.50 rows=0) (actual time=0.076..0.077 rows=15 loops=1)
        -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.001..0.015 rows=123 loops=1)
            -> Union materialize with deduplication  (cost=25.12..27.60 rows=123) (actual time=0.334..0.337 rows=123 loops=1)
                -> Table scan on BasketballTable  (cost=6.65 rows=64) (actual time=0.034..0.075 rows=64 loops=1)
                -> Table scan on FootballTable  (cost=6.15 rows=59) (actual time=0.007..0.040 rows=59 loops=1)
 |
```

1. **Indexes design/philosophy**
   i. Design 1:
      With index for Athlete only
      - ☐ CREATE INDEX Athlete_Index ON Athlete (athleteName, NOCName, discName);

```
| -> Limit: 15 row(s)  (cost=2.50 rows=0) (actual time=0.080..0.082 rows=15 loops=1)
    -> Sort: coachName, limit input to 15 row(s) per chunk  (cost=2.50 rows=0) (actual time=0.079..0.080 rows=15 loops=1)
        -> Table scan on <union temporary>  (cost=2.50 rows=0) (actual time=0.002..0.017 rows=123 loops=1)
            -> Union materialize with deduplication  (cost=2.50..2.50 rows=0) (actual time=26.246..26.250 rows=123 loops=1)
                -> Table scan on <temporary>  (actual time=0.004..0.031 rows=64 loops=1)
                    -> Aggregate using temporary table  (actual time=15.251..15.281 rows=64 loops=1)
                        -> Nested loop inner join  (cost=724.38 rows=101) (actual time=0.134..13.675 rows=1237 loops=1)
                            -> Index lookup on c using discName (discName='Basketball')  (cost=12.21 rows=74) (actual time=0.069..0.149 rows=74 loops=1)
                            -> Filter: (a.discName = 'Basketball')  (cost=4.25 rows=1) (actual time=0.030..0.181 rows=17 loops=74)
                                -> Index lookup on a using NOCName (NOCName=c.NOCName)  (cost=4.25 rows=54) (actual time=0.025..0.150 rows=276 loops=74)
                -> Table scan on <temporary>  (actual time=0.004..0.018 rows=59 loops=1)
                    -> Aggregate using temporary table  (actual time=10.744..10.762 rows=59 loops=1)
                        -> Nested loop inner join  (cost=577.58 rows=162) (actual time=0.325..9.146 rows=1439 loops=1)
                            -> Index lookup on c using discName (discName='Football')  (cost=9.78 rows=59) (actual time=0.260..0.302 rows=59 loops=1)
                            -> Filter: (a.discName = 'Football')  (cost=4.25 rows=3) (actual time=0.029..0.148 rows=24 loops=59)
                                -> Index lookup on a using NOCName (NOCName=c.NOCName)  (cost=4.25 rows=54) (actual time=0.022..0.122 rows=244 loops=59)
 |
```

   ii. Design 2:
      With index for Athlete and Coach
      - ☐ CREATE INDEX Coach_Index ON Coach (NOCName, coachName, discName, event);



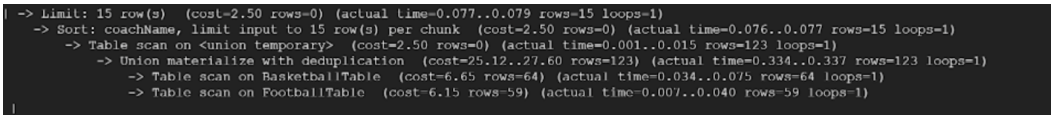
iii. Design 3:

Updated SQL Command:

```
(SELECT * FROM BasketballTable)
UNION
(SELECT * FROM FootballTable)
ORDER BY coachName
LIMIT 15;
```

With index for Athlete, Coach, FootballTable, and BasketballTable (temporary tables)

- ☐ CREATE TEMPORARY TABLE BasketballTable
  SELECT coachName, c.NOCName, c.discName,
  COUNT(athleteName) AS athleteCount
  FROM Athlete a JOIN Coach c ON (a.NOCName = c.NOCName
  AND a.discName = c.discName)
  WHERE a.discName = ‘Basketball’
  GROUP BY coachName, c.NOCName, c.discName;
- ☐ CREATE TEMPORARY TABLE FootballTable
  SELECT coachName, c.NOCName, c.discName,
  COUNT(athleteName) AS athleteCount
  FROM Athlete a JOIN Coach c ON (a.NOCName = c.NOCName
  AND a.discName = c.discName)
  WHERE a.discName = ‘Football’
  GROUP BY coachName, c.NOCName, c.discName;
- ☐ CREATE INDEX BasketballTable_Index ON BasketballTable (coachName);
- ☐ CREATE INDEX FootballTable_Index ON FootballTable (coachName);



Because our query involves a union, it does a lot of iterating over the elements of our relation to look for duplicates, so the runtime without any indexes isn't very good. First, we added an index for Athlete, and saw that it did improve the runtime a decent amount. Then, we added another index for Coach in conjunction with the one for Athlete, and together they significantly reduced the runtime for our query. For our final design, we wanted to see if creating temporary tables could speed up the performance even more, but it actually made it worse. We think this is because it takes more time and work to go through the process of creating the temporary tables and indexes for them than it's worth. So the best indexing design had indexes for Athlete and Coach.