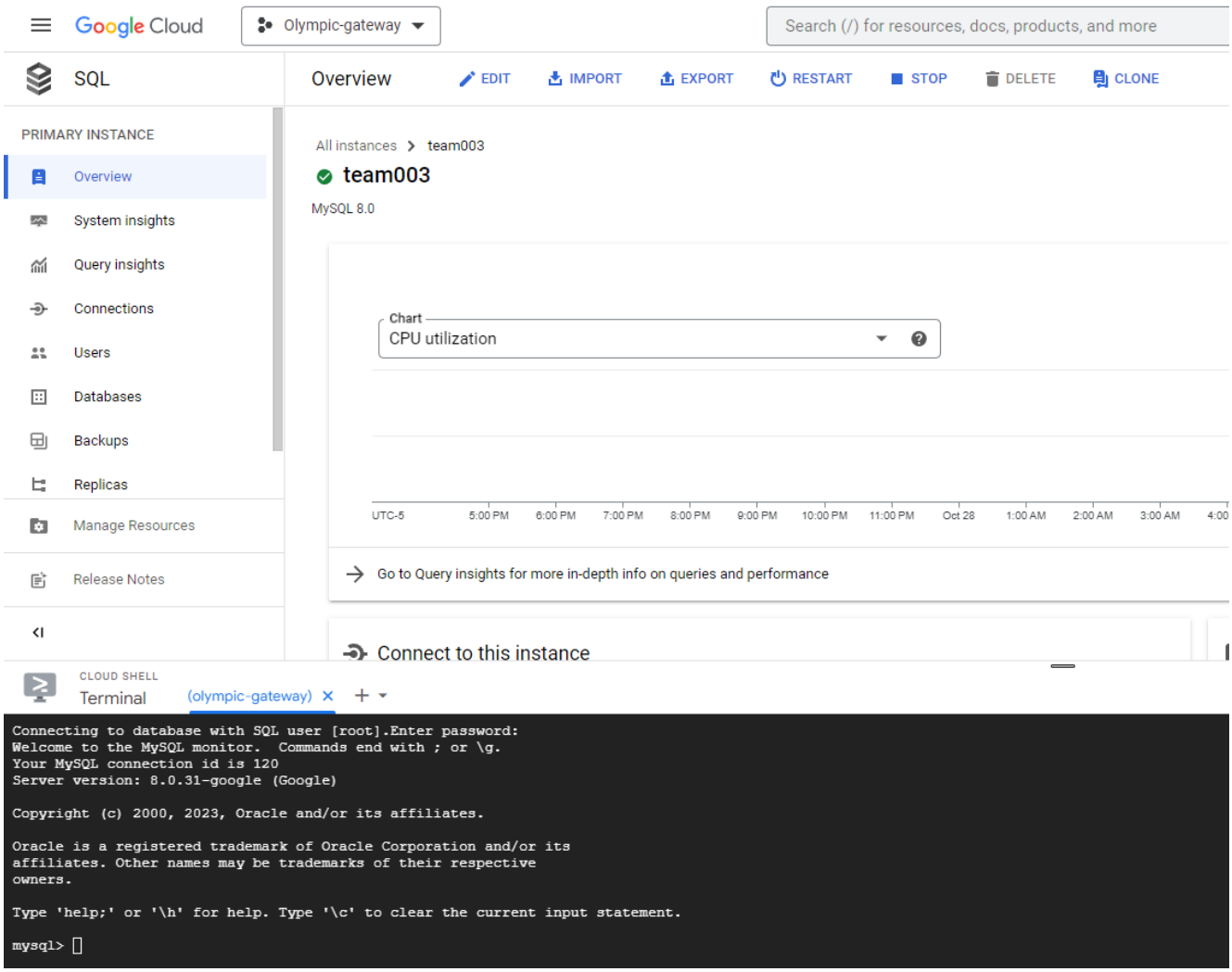
Screenshot of the connection to GCP



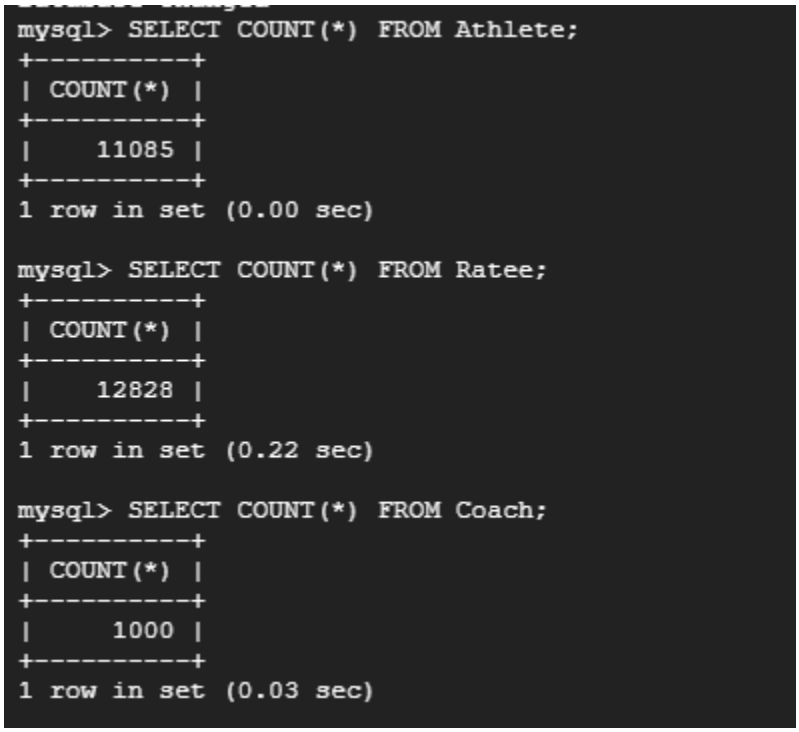
DDL commands for Create Table:

```
CREATE TABLE User (
    Email VARCHAR(255) PRIMARY KEY,
    Password VARCHAR(255),
    Username VARCHAR(255)
);

CREATE TABLE Ratee (
    RateeId INT PRIMARY KEY,
    Type VARCHAR(255),
    SumofRating INT,
    NumofRating INT
);

CREATE TABLE Comment (
    CommentId INT AUTO_INCREMENT PRIMARY KEY,
    Content VARCHAR(4095),
    Time TIMESTAMP,
    PostBy VARCHAR(255),
    Target INT,
    FOREIGN KEY (PostBy) REFERENCES User(Email),
    FOREIGN KEY (Target) REFERENCES Ratee(RateeId)
);

CREATE TABLE Rates (
    RateBy VARCHAR(255),
    Target INT,
    RatingValue INT,
    Time TIMESTAMP,
    PRIMARY KEY (RateBy, Target),
    FOREIGN KEY (RateBy) REFERENCES User(Email),
    FOREIGN KEY (Target) REFERENCES Ratee(RateeId)
);

CREATE TABLE Country (
    Country VARCHAR(255) PRIMARY KEY,
    Gold INT,
    Silver INT,
    Bronze INT,
    Ranks INT
```

```
);

CREATE TABLE Discipline (
    Discipline VARCHAR (255) PRIMARY KEY,
    MaleCount INT,
    FemaleCount INT,
    TotalPlayerCount INT
);

CREATE TABLE Athlete (
    AthleteId INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR(255),
    Discipline VARCHAR(255),
    Country VARCHAR (255),
    RateeId INT,
    FOREIGN KEY (Discipline) REFERENCES Discipline(Discipline),
    FOREIGN KEY (Country) REFERENCES Country(Country),
    FOREIGN KEY (RateeId) REFERENCES Ratee(RateeId)
);

CREATE TABLE Coach (
    CoachId INT AUTO_INCREMENT PRIMARY KEY,
    Name VARCHAR (255),
    Country VARCHAR (255),
    Discipline VARCHAR (255),
    Event VARCHAR (255),
    RateeId INT,
    FOREIGN KEY (Country) REFERENCES Country(Country),
    FOREIGN KEY (Discipline) REFERENCES Discipline(Discipline),
    FOREIGN KEY (RateeId) REFERENCES Ratee(RateeId)
);

CREATE TABLE Team (
    TeamId INT AUTO_INCREMENT PRIMARY KEY,
    Country VARCHAR (255),
    Discipline VARCHAR (255),
    Event VARCHAR (255),
    RateeId INT,
    FOREIGN KEY (Country) REFERENCES Country(Country),
    FOREIGN KEY (Discipline) REFERENCES Discipline(Discipline),
```

```
    FOREIGN KEY (RateeId) REFERENCES Ratee(RateeId)
);
```

Screenshots of at least 1000 rows in 3 of our tables



```
mysql> SELECT COUNT(*) FROM Athlete;
+----------+
| COUNT(*) |
+----------+
|    11085 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM Ratee;
+----------+
| COUNT(*) |
+----------+
|    12828 |
+----------+
1 row in set (0.22 sec)

mysql> SELECT COUNT(*) FROM Coach;
+----------+
| COUNT(*) |
+----------+
|     1000 |
+----------+
1 row in set (0.03 sec)
```

Advanced Queries

1.

For all disciplines, show the ratio between players and coaches.

```
SELECT Discipline, TotalPlayerCount/CountNum AS PlayerToCoachRatio
FROM Discipline NATURAL JOIN (SELECT COUNT(CoachID) AS CountNum, Discipline
FROM Coach GROUP BY Discipline) AS CoachCount
ORDER BY PlayerToCoachRatio DESC
LIMIT 15;
```

Output of Advanced Query 1 on our database

```
[('Water Polo', Decimal('12.1818')),
 ('Rugby Sevens', Decimal('11.8800')),
 ('Handball', Decimal('11.5862')),
 ('Football', Decimal('10.3051')),
 ('Baseball/Softball', Decimal('9.0000')),
 ('Hockey', Decimal('8.5333')),
 ('Volleyball', Decimal('6.4000')),
 ('Artistic Swimming', Decimal('1.5217')),
 ('Basketball', Decimal('0.4235'))]
```

Our output has less than 15 rows.

2.

Show the Top 15 countries with the highest Athlete_to_Medal_Ratio.

```
SELECT A.Country, COUNT(A.AthleteId) AS Number_of_Athletes, C.Gold + C.Silver +
C.Bronze AS Total_medals, COUNT(A.AthleteId) / (C.Gold + C.Silver + C.Bronze) AS
Athlete_to_Medal_Ratio
FROM  Athlete A JOIN Country C ON A.Country = C.Country
GROUP BY A.Country
ORDER BY Athlete_to_Medal_Ratio DESC
LIMIT 15;
```

Output of Advanced Query 2 on our database

```
[('Argentina', 180, 3, Decimal('60.0000')),
 ('South Africa', 171, 3, Decimal('57.0000')),
 ('Morocco', 48, 1, Decimal('48.0000')),
 ('Mexico', 155, 4, Decimal('38.7500')),
 ('Lithuania', 37, 1, Decimal('37.0000')),
 ('Puerto Rico', 35, 1, Decimal('35.0000')),
 ('Saudi Arabia', 32, 1, Decimal('32.0000')),
 ('Bahrain', 31, 1, Decimal('31.0000')),
 ('Nigeria', 59, 2, Decimal('29.5000')),
 ('Ireland', 116, 4, Decimal('29.0000')),
 ("Côte d'Ivoire", 29, 1, Decimal('29.0000')),
 ('Tunisia', 57, 2, Decimal('28.5000')),
 ('Romania', 99, 4, Decimal('24.7500')),
 ('Finland', 45, 2, Decimal('22.5000')),
 ('Egypt', 133, 6, Decimal('22.1667'))]
```

Indexing Analysis

For advanced query 1:

Indexing design 1:

EXPLAIN ANALYZE before indexing:

```
mysql> EXPLAIN ANALYZE SELECT Discipline, TotalPlayerCount/CountNum AS PlayerToCoachRatio FROM Discipline NATURAL JOIN (SELECT COUNT(CoachID) AS CountNum, Discipline FROM Co
ach GROUP BY Discipline) AS CoachCount ORDER BY PlayerToCoachRatio DESC LIMIT 15;
+--------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                                |
+--------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=0.841..0.843 rows=9 loops=1)
    -> Sort: PlayerToCoachRatio DESC, limit input to 15 row(s) per chunk  (actual time=0.841..0.841 rows=9 loops=1)
        -> Stream results  (cost=465.00 rows=1000) (actual time=0.782..0.820 rows=9 loops=1)
            -> Nested loop inner join  (cost=465.00 rows=1000) (actual time=0.769..0.792 rows=9 loops=1)
                -> Filter: (CoachCount.Discipline is not null)  (cost=301.31..115.00 rows=1000) (actual time=0.730..0.733 rows=9 loops=1)
                    -> Table scan on CoachCount  (cost=301.51..316.50 rows=1000) (actual time=0.729..0.731 rows=9 loops=1)
                        -> Materialize  (cost=301.50..301.50 rows=1000) (actual time=0.726..0.726 rows=9 loops=1)
                            -> Group aggregate: count(Coach.CoachId)  (cost=201.50 rows=1000) (actual time=0.096..0.639 rows=9 loops=1)
                                -> Covering index scan on Coach using Discipline  (cost=101.50 rows=1000) (actual time=0.051..0.343 rows=1000 loops=1)
                -> Single-row index lookup on Discipline using PRIMARY (Discipline=CoachCount.Discipline)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=9)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

CREATE INDEX command:

The first index we choose is Discipline, as in our sub-query we group by the Discipline and probably an index on Discipline will improve the overall performance of our query.

```
mysql> CREATE INDEX idx_disc ON Coach(Discipline);
Query OK, 0 rows affected (0.18 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

EXPLAIN ANALYZE after indexing:

```
mysql> EXPLAIN ANALYZE SELECT Discipline, TotalPlayerCount/CountNum AS PlayerToCoachRatio FROM Discipline NATURAL JOIN (SELECT COUNT(CoachID) AS CountNum, Discipline FROM
 Coach GROUP BY Discipline) AS CoachCount ORDER BY PlayerToCoachRatio DESC LIMIT 15;
+------------------+
| EXPLAIN
      |
+------------------+
| -> Limit: 15 row(s)  (actual time=0.689..0.690 rows=9 loops=1)
    -> Sort: PlayerToCoachRatio DESC, limit input to 15 row(s) per chunk  (actual time=0.688..0.689 rows=9 loops=1)
        -> Stream results  (cost=465.00 rows=1000) (actual time=0.646..0.672 rows=9 loops=1)
            -> Nested loop inner join  (cost=465.00 rows=1000) (actual time=0.640..0.661 rows=9 loops=1)
                -> Filter: (CoachCount.Discipline is not null)  (cost=301.31..115.00 rows=1000) (actual time=0.624..0.627 rows=9 loops=1)
                    -> Table scan on CoachCount  (cost=301.51..316.50 rows=1000) (actual time=0.623..0.625 rows=9 loops=1)
                        -> Materialize  (cost=301.50..301.50 rows=1000) (actual time=0.621..0.621 rows=9 loops=1)
                            -> Group aggregate: count(Coach.CoachId)  (cost=201.50 rows=1000) (actual time=0.132..0.605 rows=9 loops=1)
                                -> Covering index scan on Coach using idx_disc  (cost=101.50 rows=1000) (actual time=0.043..0.314 rows=1000 loops=1)
                -> Single-row index lookup on Discipline using PRIMARY (Discipline=CoachCount.Discipline)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=9)
 |
+------------------+
1 row in set (0.00 sec)
```

As we can see, by creating a time on Coach(Discipline), the actual time reduced from 0.096 to 0.043. The execution time reduced from 0.01 sec to 0.00 sec.

Indexing design 2:

EXPLAIN ANALYZE before indexing:

```
mysql> EXPLAIN ANALYZE SELECT Discipline, TotalPlayerCount/CountNum AS PlayerToCoachRatio FROM Discipline NATURAL JOIN (SELECT COUNT(CoachID) AS CountNum, Discipline FROM Co
ach GROUP BY Discipline) AS CoachCount ORDER BY PlayerToCoachRatio DESC LIMIT 15;
+------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                        |
+------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=0.841..0.843 rows=9 loops=1)
    -> Sort: PlayerToCoachRatio DESC, limit input to 15 row(s) per chunk  (actual time=0.841..0.841 rows=9 loops=1)
        -> Stream results  (cost=465.00 rows=1000) (actual time=0.782..0.820 rows=9 loops=1)
            -> Nested loop inner join  (cost=465.00 rows=1000) (actual time=0.769..0.792 rows=9 loops=1)
                -> Filter: (CoachCount.Discipline is not null)  (cost=301.31..115.00 rows=1000) (actual time=0.730..0.733 rows=9 loops=1)
                    -> Table scan on CoachCount  (cost=301.51..316.50 rows=1000) (actual time=0.729..0.731 rows=9 loops=1)
                        -> Materialize  (cost=301.50..301.50 rows=1000) (actual time=0.726..0.726 rows=9 loops=1)
                            -> Group aggregate: count(Coach.CoachId)  (cost=201.50 rows=1000) (actual time=0.096..0.639 rows=9 loops=1)
                                -> Covering index scan on Coach using Discipline  (cost=101.50 rows=1000) (actual time=0.051..0.343 rows=1000 loops=1)
                -> Single-row index lookup on Discipline using PRIMARY (Discipline=CoachCount.Discipline)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=9)
 |
+------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

CREATE INDEX command:

We choose to create an index on Event on the Coach Table. We choose this because we want to check whether an unused key will affect the performance of our advanced query.

```
mysql> CREATE INDEX idx_event ON Coach(Event);
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

EXPLAIN ANALYZE after indexing:

```
mysql> EXPLAIN ANALYZE SELECT Discipline, TotalPlayerCount/CountNum AS PlayerToCoachRatio FROM Discipline NATURAL JOIN (SELECT COUNT(CoachID) AS CountNum, Discipline FROM Co
ach GROUP BY Discipline) AS CoachCount ORDER BY PlayerToCoachRatio DESC LIMIT 15;
+------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                        |
+------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=0.887..0.889 rows=9 loops=1)
    -> Sort: PlayerToCoachRatio DESC, limit input to 15 row(s) per chunk  (actual time=0.887..0.888 rows=9 loops=1)
        -> Stream results  (cost=465.00 rows=1000) (actual time=0.823..0.855 rows=9 loops=1)
            -> Nested loop inner join  (cost=465.00 rows=1000) (actual time=0.806..0.833 rows=9 loops=1)
                -> Filter: (CoachCount.Discipline is not null)  (cost=301.31..115.00 rows=1000) (actual time=0.770..0.774 rows=9 loops=1)
                    -> Table scan on CoachCount  (cost=301.51..316.50 rows=1000) (actual time=0.769..0.771 rows=9 loops=1)
                        -> Materialize  (cost=301.50..301.50 rows=1000) (actual time=0.761..0.761 rows=9 loops=1)
                            -> Group aggregate: count(Coach.CoachId)  (cost=201.50 rows=1000) (actual time=0.262..0.735 rows=9 loops=1)
                                -> Covering index scan on Coach using idx_disc  (cost=101.50 rows=1000) (actual time=0.207..0.466 rows=1000 loops=1)
                -> Single-row index lookup on Discipline using PRIMARY (Discipline=CoachCount.Discipline)  (cost=0.25 rows=1) (actual time=0.006..0.006 rows=1 loops=9)
 |
+------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

There is no improvement on this indexing because we create an index on Coach(Event) which is not queried during the Advanced Query.

Indexing design 3:

EXPLAIN ANALYZE before indexing:

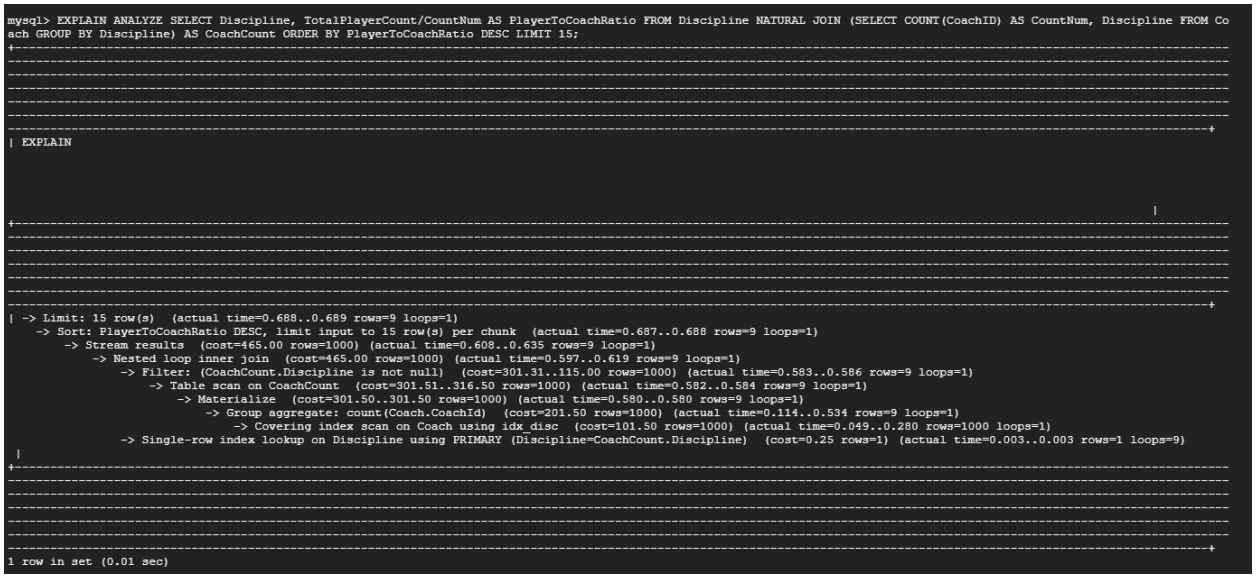
```
mysql> EXPLAIN ANALYZE SELECT Discipline, TotalPlayerCount/CountNum AS PlayerToCoachRatio FROM Discipline NATURAL JOIN (SELECT COUNT(CoachID) AS CountNum, Discipline FROM Co
ach GROUP BY Discipline) AS CoachCount ORDER BY PlayerToCoachRatio DESC LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                                        |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=0.841..0.843 rows=9 loops=1)
    -> Sort: PlayerToCoachRatio DESC, limit input to 15 row(s) per chunk  (actual time=0.841..0.841 rows=9 loops=1)
        -> Stream results  (cost=465.00 rows=1000) (actual time=0.782..0.820 rows=9 loops=1)
            -> Nested loop inner join  (cost=465.00 rows=1000) (actual time=0.769..0.792 rows=9 loops=1)
                -> Filter: (CoachCount.Discipline is not null)  (cost=301.31..115.00 rows=1000) (actual time=0.730..0.733 rows=9 loops=1)
                    -> Table scan on CoachCount  (cost=301.51..316.50 rows=1000) (actual time=0.729..0.731 rows=9 loops=1)
                        -> Materialize  (cost=301.50..301.50 rows=1000) (actual time=0.726..0.726 rows=9 loops=1)
                            -> Group aggregate: count(Coach.CoachId)  (cost=201.50 rows=1000) (actual time=0.096..0.639 rows=9 loops=1)
                                -> Covering index scan on Coach using Discipline  (cost=101.50 rows=1000) (actual time=0.051..0.343 rows=1000 loops=1)
                -> Single-row index lookup on Discipline using PRIMARY (Discipline=CoachCount.Discipline)  (cost=0.25 rows=1) (actual time=0.005..0.005 rows=1 loops=9)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)
```

CREATE INDEX command:

We create an index on TotalPlayerCount on Discipline table because this key is used for calculating our final PlayToCoachRatio.

```
mysql> CREATE INDEX idx_pcount ON Discipline(TotalPlayerCount);
Query OK, 0 rows affected (0.26 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

EXPLAIN ANALYZE after indexing:



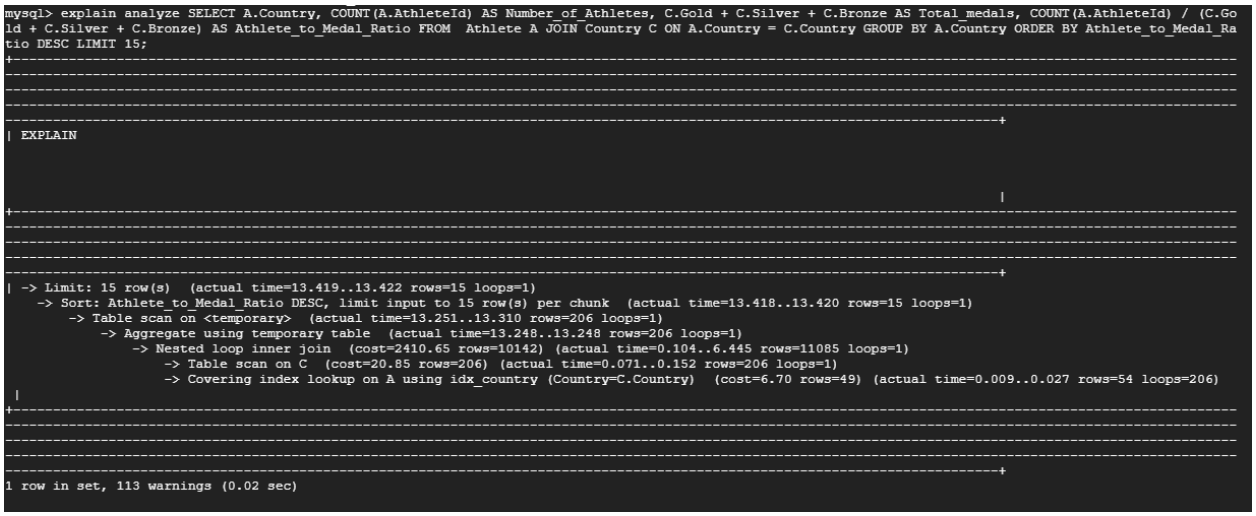
As we can see, this time we created the pcount index, and the Table scan on CoachCount improved from 0.726 to 0.582.

For advanced query 2:

Indexing design 1:

EXPLAIN ANALYZE before indexing:

No indices created



CREATE INDEX command:

```
mysql> create index idx_gold on Country (Gold);
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc Country
    -> ;
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| Country | varchar(255) | NO   | PRI | NULL    |       |
| Gold    | int          | YES  | MUL | NULL    |       |
| Silver  | int          | YES  |     | NULL    |       |
| Bronze  | int          | YES  |     | NULL    |       |
| Ranks   | int          | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+
5 rows in set (0.01 sec)

mysql> create index idx_silver on COuntry (Silver);
ERROR 1146 (42S02): Table 'db.COuntry' doesn't exist
mysql> create index idx_silver on Country (Silver);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> create index idx_bronze on Country (Bronze);
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

EXPLAIN ANALYZE after indexing:

```
mysql> explain analyze SELECT A.Country, COUNT(A.AthleteId) AS Number_of_Athletes, C.Gold + C.Silver + C.Bronze AS Total_medals, COUNT(A.AthleteId) / (C.Go
ld + C.Silver + C.Bronze) AS Athlete_to_Medal_Ratio
    -> FROM  Athlete A JOIN Country C ON A.Country = C.Country
    -> GROUP BY A.Country
    -> ORDER BY Athlete_to_Medal_Ratio DESC
    -> LIMIT 15;
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN


                                                                                                                                                |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=14.373..14.376 rows=15 loops=1)
    -> Sort: Athlete_to_Medal_Ratio DESC, limit input to 15 row(s) per chunk  (actual time=14.372..14.374 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=14.214..14.273 rows=206 loops=1)
            -> Aggregate using temporary table  (actual time=14.212..14.212 rows=206 loops=1)
                -> Nested loop inner join  (cost=2410.65 rows=10142) (actual time=0.070..7.158 rows=11085 loops=1)
                    -> Table scan on C  (cost=20.85 rows=206) (actual time=0.047..0.140 rows=206 loops=1)
                    -> Covering index lookup on A using idx_country (Country=C.Country)  (cost=6.70 rows=49) (actual time=0.010..0.029 rows=54 loops=206)
 |
+-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set, 113 warnings (0.02 sec)
```

Indexing design 2:

EXPLAIN ANALYZE before indexing:

Drop the indices created in design 1

Use the same basecase reference as in design 1

```
mysql> explain analyze SELECT A.Country, COUNT(A.AthleteId) AS Number_of_Athletes, C.Gold + C.Silver + C.Bronze AS Total_medals, COUNT(A.AthleteId) / (C.Go
ld + C.Silver + C.Bronze) AS Athlete_to_Medal_Ratio FROM  Athlete A JOIN Country C ON A.Country = C.Country GROUP BY A.Country ORDER BY Athlete_to_Medal_Ra
tio DESC LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                                  |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=13.419..13.422 rows=15 loops=1)
    -> Sort: Athlete_to_Medal_Ratio DESC, limit input to 15 row(s) per chunk  (actual time=13.418..13.420 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=13.251..13.310 rows=206 loops=1)
            -> Aggregate using temporary table  (actual time=13.248..13.248 rows=206 loops=1)
                -> Nested loop inner join  (cost=2410.65 rows=10142) (actual time=0.104..6.445 rows=11085 loops=1)
                    -> Table scan on C  (cost=20.85 rows=206) (actual time=0.071..0.152 rows=206 loops=1)
                    -> Covering index lookup on A using idx_country (Country=C.Country)  (cost=6.70 rows=49) (actual time=0.009..0.027 rows=54 loops=206)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set, 113 warnings (0.02 sec)
```

CREATE INDEX command:

```
mysql> create index idx_name on Athlete (Name);
Query OK, 0 rows affected (0.17 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

EXPLAIN ANALYZE after indexing:

```
mysql> explain analyze SELECT A.Country, COUNT(A.AthleteId) AS Number_of_Athletes, C.Gold + C.Silver + C.Bronze AS Total_medals, COUNT(A.AthleteId) / (C.Go
ld + C.Silver + C.Bronze) AS Athlete_to_Medal_Ratio
    -> FROM  Athlete A JOIN Country C ON A.Country = C.Country
    -> GROUP BY A.Country
    -> ORDER BY Athlete_to_Medal_Ratio DESC
    -> LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                                  |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=14.534..14.536 rows=15 loops=1)
    -> Sort: Athlete_to_Medal_Ratio DESC, limit input to 15 row(s) per chunk  (actual time=14.533..14.534 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=14.363..14.432 rows=206 loops=1)
            -> Aggregate using temporary table  (actual time=14.360..14.360 rows=206 loops=1)
                -> Nested loop inner join  (cost=2410.65 rows=10142) (actual time=0.094..7.090 rows=11085 loops=1)
                    -> Table scan on C  (cost=20.85 rows=206) (actual time=0.069..0.337 rows=206 loops=1)
                    -> Covering index lookup on A using idx_country (Country=C.Country)  (cost=6.70 rows=49) (actual time=0.009..0.029 rows=54 loops=206)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set, 113 warnings (0.01 sec)
```

Indexing design 3:

EXPLAIN ANALYZE before indexing:

Drop the indices created in design 2

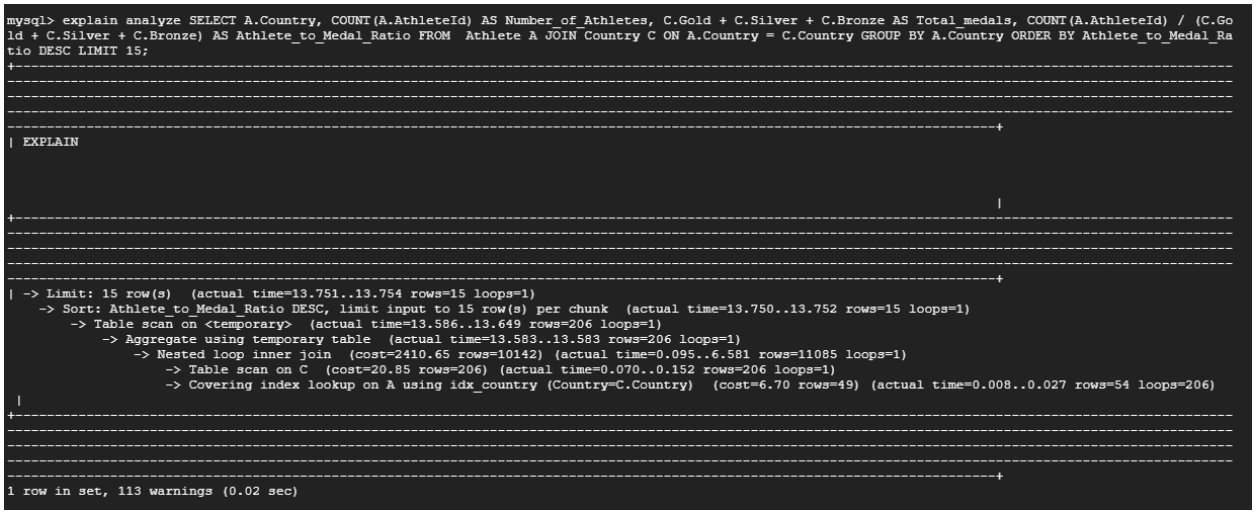
Use the same basecase reference as in design 1

```
mysql> explain analyze SELECT A.Country, COUNT(A.AthleteId) AS Number_of_Athletes, C.Gold + C.Silver + C.Bronze AS Total_medals, COUNT(A.AthleteId) / (C.Go
ld + C.Silver + C.Bronze) AS Athlete_to_Medal_Ratio FROM  Athlete A JOIN Country C ON A.Country = C.Country GROUP BY A.Country ORDER BY Athlete_to_Medal_Ra
tio DESC LIMIT 15;
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN                                                                                                                                                  |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=13.419..13.422 rows=15 loops=1)
    -> Sort: Athlete_to_Medal_Ratio DESC, limit input to 15 row(s) per chunk  (actual time=13.418..13.420 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=13.251..13.310 rows=206 loops=1)
            -> Aggregate using temporary table  (actual time=13.248..13.248 rows=206 loops=1)
                -> Nested loop inner join  (cost=2410.65 rows=10142) (actual time=0.104..6.445 rows=11085 loops=1)
                    -> Table scan on C  (cost=20.85 rows=206) (actual time=0.071..0.152 rows=206 loops=1)
                    -> Covering index lookup on A using idx_country (Country=C.Country)  (cost=6.70 rows=49) (actual time=0.009..0.027 rows=54 loops=206)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------+
1 row in set, 113 warnings (0.02 sec)
```

CREATE INDEX command:

```
mysql> create index idx_ranks on Country (Ranks);
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

EXPLAIN ANALYZE after indexing:



In all three designs, we do not see significant differences in performance. All queries run into table scan. Since our query relies only on primary keys and calculated values on the fly, adding indices does not improve the performance.
When designing indices, we first identify the columns involved in calculations and join operations, then gradually add indices into the tables.
The first design adds indices for each gold, silver, and bronze column in the Country table. These columns participate in calculating the final value, but do not participate in querying. So the performance does not get better.
The second design adds an index for the name column in the Athletes table, making all of its columns indexed. But still, since the name column does not participate in querying, the performance does not get better.
The third design adds an index for all columns in both Athlete and Country tables. Still, since only the primary keys are involved in the join operation, the performance does not get better.