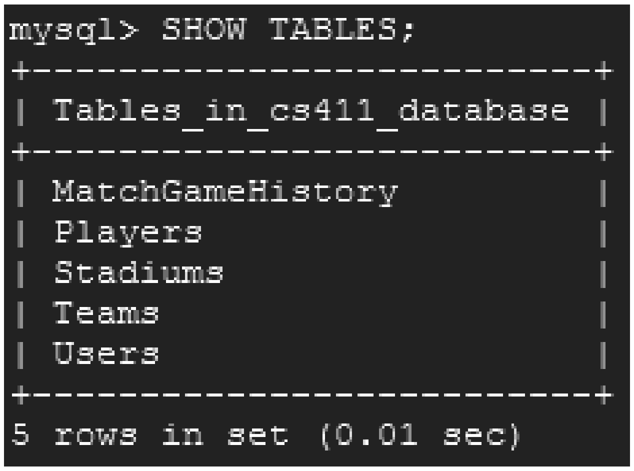
Part One

Tables implemented



```
mysql> SHOW TABLES;
+-------------------------+
| Tables_in_cs411_database |
+-------------------------+
| MatchGameHistory        |
| Players                 |
| Stadiums                |
| Teams                   |
| Users                   |
+-------------------------+
5 rows in set (0.01 sec)
```

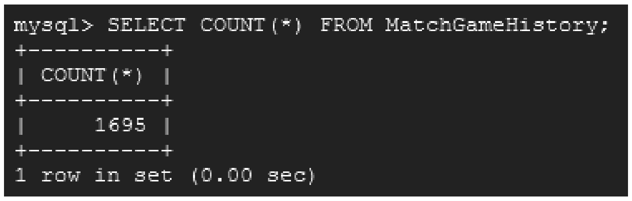
DDL Commands

```
CREATE TABLE MatchGameHistory (
	GameId INT,
	Attendance INT,
	HomeTeam VARCHAR(255),
	AwayTeam VARCHAR(255),
	HomeTeamGoals INT,
	AwayTeamGoals INT,
	Stadium VARCHAR(255),
	PRIMARY KEY(GameId),
	FOREIGN KEY(HomeTeam) REFERENCES Teams(TeamName),
	FOREIGN KEY(AwayTeam) REFERENCES Teams(TeamName),
	FOREIGN KEY(Stadium) REFERENCES Stadiums(StadiumName)
);
```

```
CREATE TABLE Players (
	PlayerId INT,
	PlayerName VARCHAR(255),
	Team VARCHAR(255),
	Appearances INT,
	GoalsScored INT,
	Cards INT,
	CleanSheets INT,
	Assists INT,
        Position VARCHAR(255),
        PRIMARY KEY(PlayerId),
        FOREIGN KEY(Team) REFERENCES Teams(TeamName)
);

CREATE TABLE Stadiums (
        StadiumName VARCHAR(255),
        Capacity INT,
        SurfaceType VARCHAR(255),
        Location VARCHAR(255),
        PRIMARY KEY(StadiumName)
);

CREATE TABLE Teams (
        TeamName VARCHAR(255),
        GoalDifference INT,
        Wins INT,
        Losses INT,
        PRIMARY KEY(TeamName)
);

CREATE TABLE Users (
        UserId VARCHAR(50),
        UserName VARCHAR(50),
        Email VARCHAR(100),
        PRIMARY KEY(UserId)
);
```

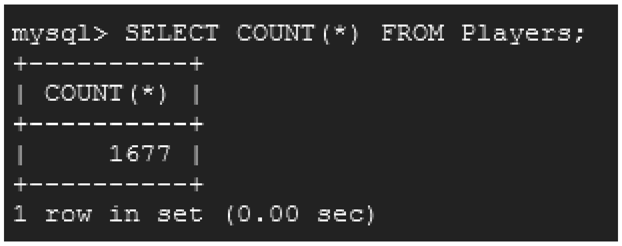
SELECT COUNT(*) FROM Table_Name

<u>MatchGameHistory</u>



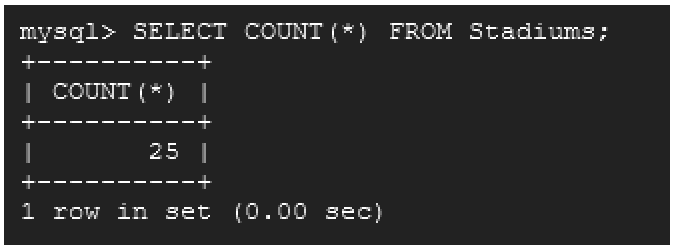
```
mysql> SELECT COUNT(*) FROM MatchGameHistory;
+----------+
| COUNT(*) |
+----------+
|     1695 |
+----------+
1 row in set (0.00 sec)
```

Players



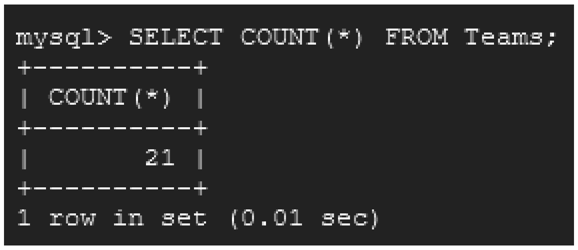
```
mysql> SELECT COUNT(*) FROM Players;
+----------+
| COUNT(*) |
+----------+
|     1677 |
+----------+
1 row in set (0.00 sec)
```

Stadiums



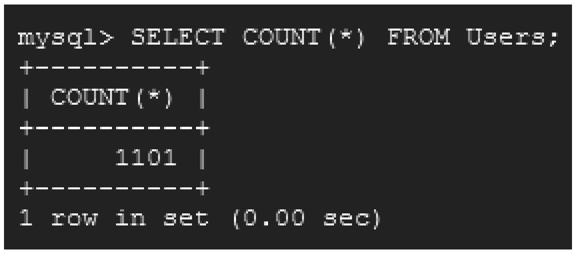
```
mysql> SELECT COUNT(*) FROM Stadiums;
+----------+
| COUNT(*) |
+----------+
|       25 |
+----------+
1 row in set (0.00 sec)
```

Teams



```
mysql> SELECT COUNT(*) FROM Teams;
+----------+
| COUNT(*) |
+----------+
|       21 |
+----------+
1 row in set (0.01 sec)
```

Users

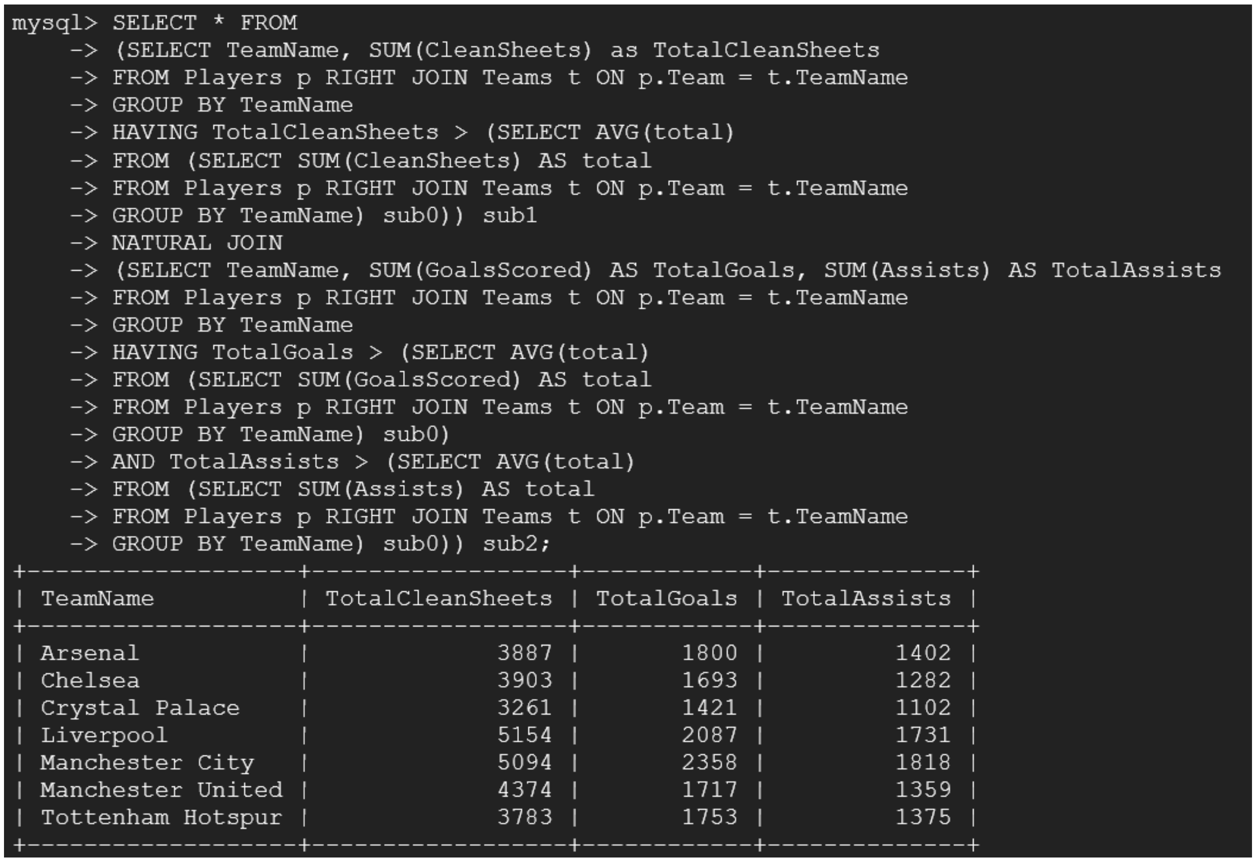


```
mysql> SELECT COUNT(*) FROM Users;
+----------+
| COUNT(*) |
+----------+
|     1101 |
+----------+
1 row in set (0.00 sec)
```

Queries

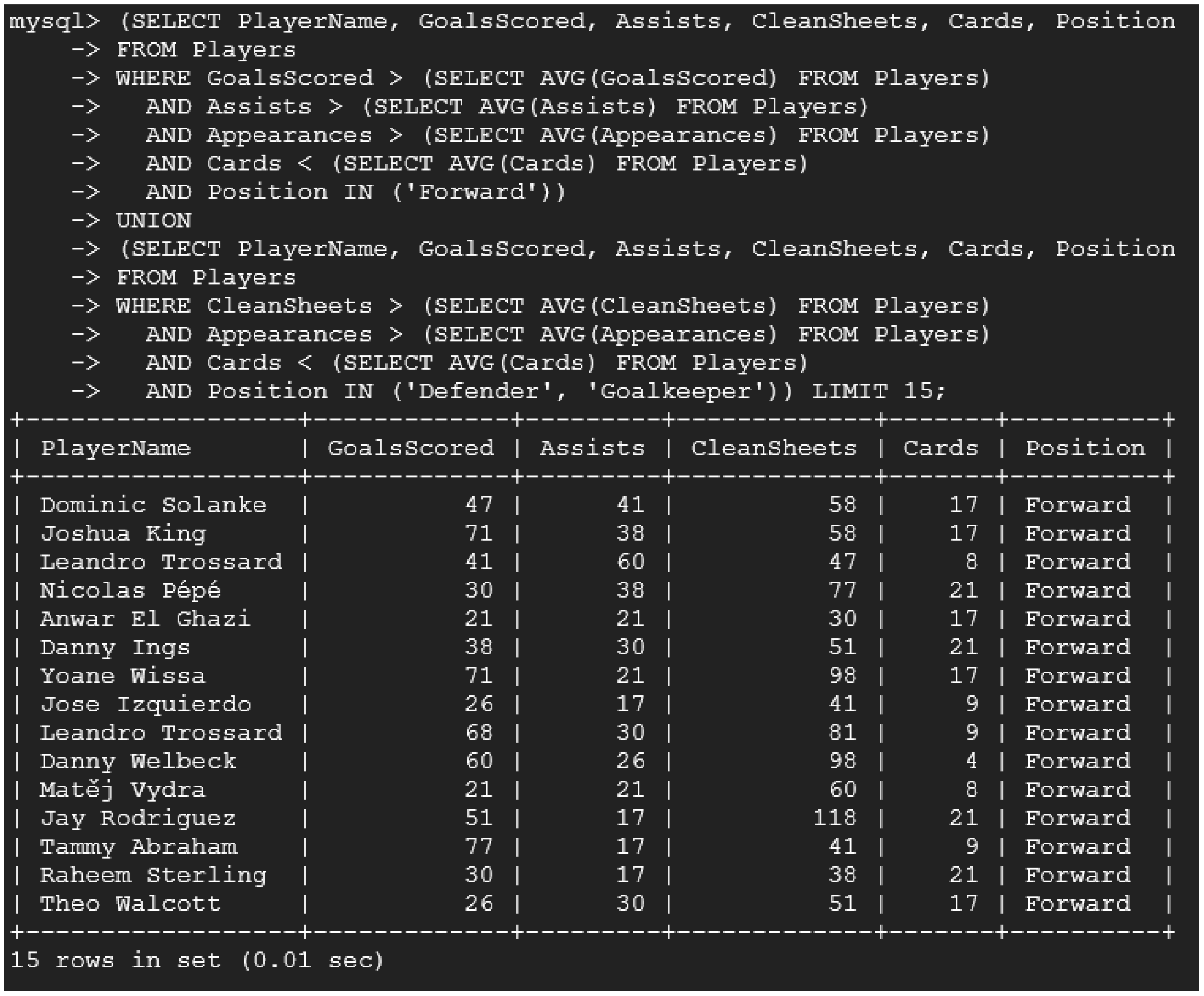
These queries will likely be used in an algorithm we will design for stage 4 that weights these results to select teams/players for fantasy football.

1. **Selecting teams that are above average in attacking and defending capability (output is only 7 rows)**



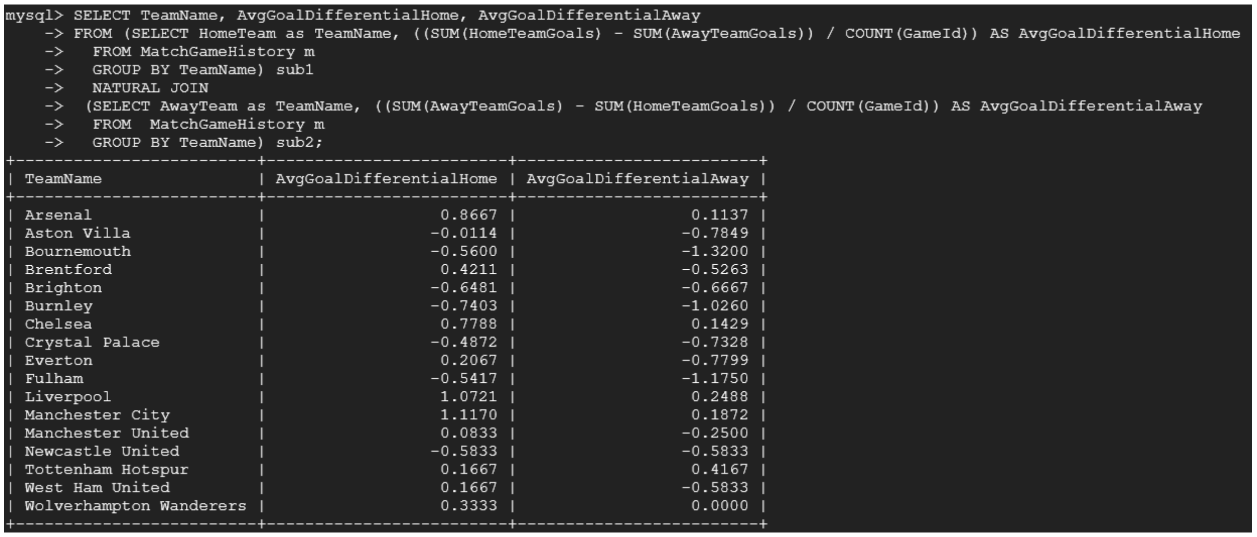
```
mysql> SELECT * FROM
    -> (SELECT TeamName, SUM(CleanSheets) as TotalCleanSheets
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalCleanSheets > (SELECT AVG(total)
    -> FROM (SELECT SUM(CleanSheets) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub1
    -> NATURAL JOIN
    -> (SELECT TeamName, SUM(GoalsScored) AS TotalGoals, SUM(Assists) AS TotalAssists
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalGoals > (SELECT AVG(total)
    -> FROM (SELECT SUM(GoalsScored) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)
    -> AND TotalAssists > (SELECT AVG(total)
    -> FROM (SELECT SUM(Assists) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub2;
+-------------------+------------------+------------+--------------+
| TeamName          | TotalCleanSheets | TotalGoals | TotalAssists |
+-------------------+------------------+------------+--------------+
| Arsenal           |             3887 |       1800 |         1402 |
| Chelsea           |             3903 |       1693 |         1282 |
| Crystal Palace    |             3261 |       1421 |         1102 |
| Liverpool         |             5154 |       2087 |         1731 |
| Manchester City   |             5094 |       2358 |         1818 |
| Manchester United |             4374 |       1717 |         1359 |
| Tottenham Hotspur |             3783 |       1753 |         1375 |
+-------------------+------------------+------------+--------------+
```

2. Union of above average forward and defensive players that don’t get carded often



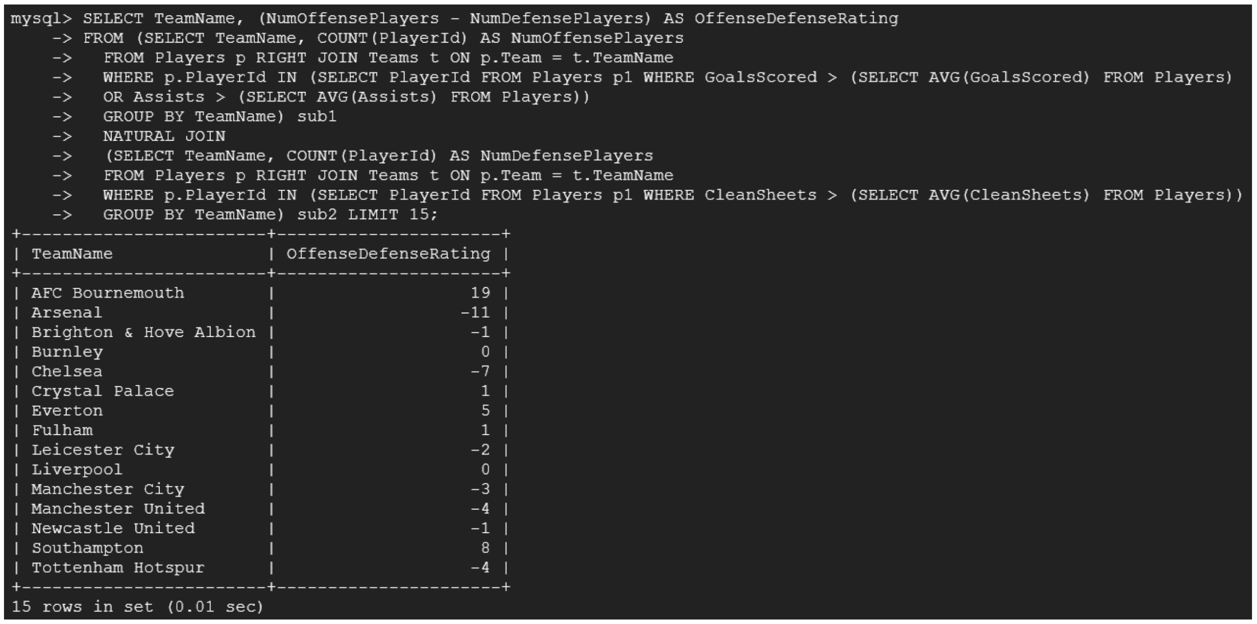
```
mysql> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   AND Assists > (SELECT AVG(Assists) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Forward'))
    -> UNION
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Defender', 'Goalkeeper')) LIMIT 15;
+------------------+-------------+---------+-------------+-------+----------+
| PlayerName       | GoalsScored | Assists | CleanSheets | Cards | Position |
+------------------+-------------+---------+-------------+-------+----------+
| Dominic Solanke  |          47 |      41 |          58 |    17 | Forward  |
| Joshua King      |          71 |      38 |          58 |    17 | Forward  |
| Leandro Trossard |          41 |      60 |          47 |     8 | Forward  |
| Nicolas Pépé     |          30 |      38 |          77 |    21 | Forward  |
| Anwar El Ghazi   |          21 |      21 |          30 |    17 | Forward  |
| Danny Ings       |          38 |      30 |          51 |    21 | Forward  |
| Yoane Wissa      |          71 |      21 |          98 |    17 | Forward  |
| Jose Izquierdo   |          26 |      17 |          41 |     9 | Forward  |
| Leandro Trossard |          68 |      30 |          81 |     9 | Forward  |
| Danny Welbeck    |          60 |      26 |          98 |     4 | Forward  |
| Matěj Vydra      |          21 |      21 |          60 |     8 | Forward  |
| Jay Rodriguez    |          51 |      17 |         118 |    21 | Forward  |
| Tammy Abraham    |          77 |      17 |          41 |     9 | Forward  |
| Raheem Sterling  |          30 |      17 |          38 |    21 | Forward  |
| Theo Walcott     |          26 |      30 |          51 |    17 | Forward  |
+------------------+-------------+---------+-------------+-------+----------+
15 rows in set (0.01 sec)
```

3. **Comparison of teams at home vs away**



```
mysql> SELECT TeamName, AvgGoalDifferentialHome, AvgGoalDifferentialAway
    -> FROM (SELECT HomeTeam as TeamName, ((SUM(HomeTeamGoals) - SUM(AwayTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialHome
    ->   FROM MatchGameHistory m
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->  (SELECT AwayTeam as TeamName, ((SUM(AwayTeamGoals) - SUM(HomeTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialAway
    ->   FROM  MatchGameHistory m
    ->   GROUP BY TeamName) sub2;
+-------------------------+-------------------------+-------------------------+
| TeamName                | AvgGoalDifferentialHome | AvgGoalDifferentialAway |
+-------------------------+-------------------------+-------------------------+
| Arsenal                 |                  0.8667 |                  0.1137 |
| Aston Villa             |                 -0.0114 |                 -0.7849 |
| Bournemouth             |                 -0.5600 |                 -1.3200 |
| Brentford               |                  0.4211 |                 -0.5263 |
| Brighton                |                 -0.6481 |                 -0.6667 |
| Burnley                 |                 -0.7403 |                 -1.0260 |
| Chelsea                 |                  0.7788 |                  0.1429 |
| Crystal Palace          |                 -0.4872 |                 -0.7328 |
| Everton                 |                  0.2067 |                 -0.7799 |
| Fulham                  |                 -0.5417 |                 -1.1750 |
| Liverpool               |                  1.0721 |                  0.2488 |
| Manchester City         |                  1.1170 |                  0.1872 |
| Manchester United       |                  0.0833 |                 -0.2500 |
| Newcastle United        |                 -0.5833 |                 -0.5833 |
| Tottenham Hotspur       |                  0.1667 |                  0.4167 |
| West Ham United         |                  0.1667 |                 -0.5833 |
| Wolverhampton Wanderers |                  0.3333 |                  0.0000 |
+-------------------------+-------------------------+-------------------------+
```

4. **Characterizing teams as either attack-oriented or defense-oriented (more attacking players is a more positive, more defense players is more negative)**



```
mysql> SELECT TeamName, (NumOffensePlayers - NumDefensePlayers) AS OffenseDefenseRating
    -> FROM (SELECT TeamName, COUNT(PlayerId) AS NumOffensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   OR Assists > (SELECT AVG(Assists) FROM Players))
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->   (SELECT TeamName, COUNT(PlayerId) AS NumDefensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players))
    ->   GROUP BY TeamName) sub2 LIMIT 15;
+------------------------+----------------------+
| TeamName               | OffenseDefenseRating |
+------------------------+----------------------+
| AFC Bournemouth        |                   19 |
| Arsenal                |                  -11 |
| Brighton & Hove Albion |                   -1 |
| Burnley                |                    0 |
| Chelsea                |                   -7 |
| Crystal Palace         |                    1 |
| Everton                |                    5 |
| Fulham                 |                    1 |
| Leicester City         |                   -2 |
| Liverpool              |                    0 |
| Manchester City        |                   -3 |
| Manchester United      |                   -4 |
| Newcastle United       |                   -1 |
| Southampton            |                    8 |
| Tottenham Hotspur      |                   -4 |
+------------------------+----------------------+
15 rows in set (0.01 sec)
```

Part Two: Indexing

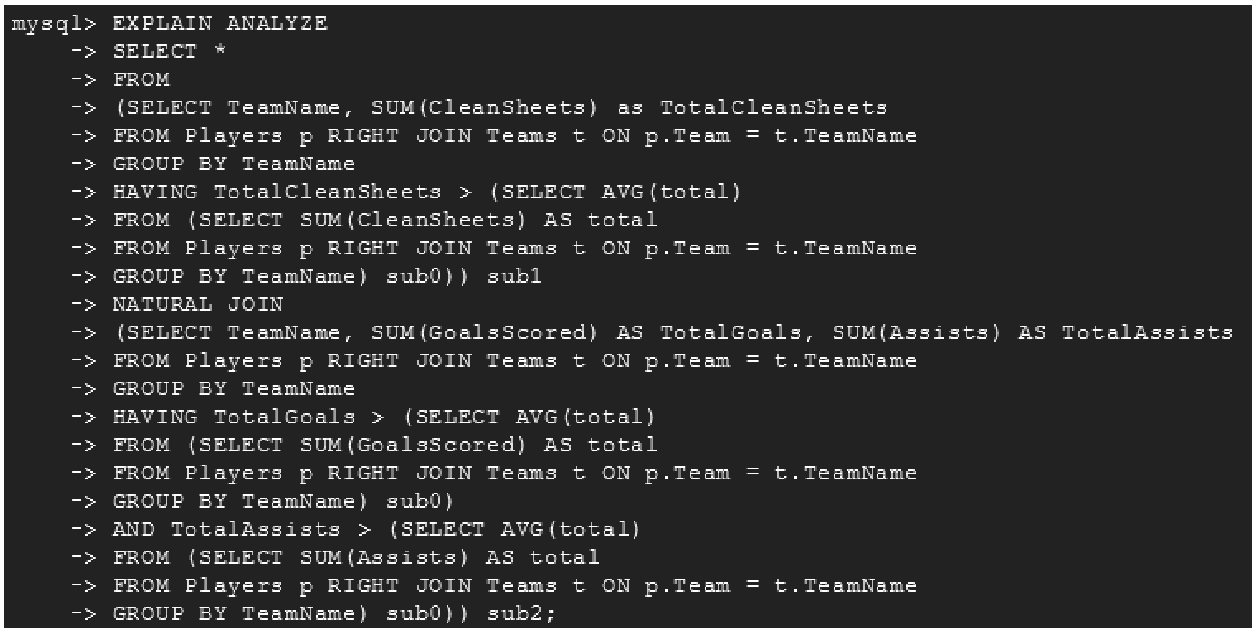
Note: Queries in this section correspond with the numbering from above.

For each query, we start from a state of no indices on any table to observe which attributes are most important to index.

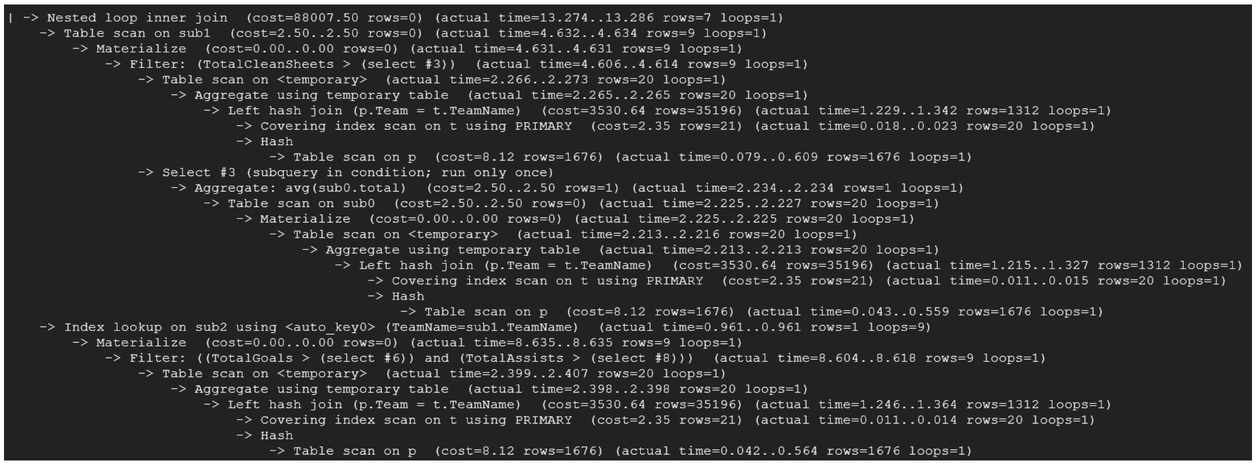
Each indexing scheme is tested in isolation from other indexing schemes. The indexing scheme we will decide on shall be the combination of indices (including no indices) that returned the lowest cost value.

Query 1

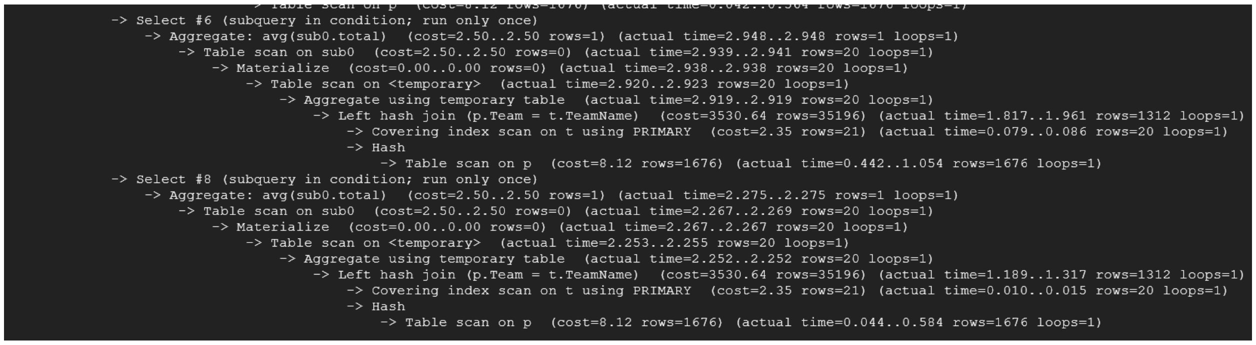
Before Indexing



```
mysql> EXPLAIN ANALYZE
    -> SELECT *
    -> FROM
    -> (SELECT TeamName, SUM(CleanSheets) as TotalCleanSheets
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalCleanSheets > (SELECT AVG(total)
    -> FROM (SELECT SUM(CleanSheets) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub1
    -> NATURAL JOIN
    -> (SELECT TeamName, SUM(GoalsScored) AS TotalGoals, SUM(Assists) AS TotalAssists
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalGoals > (SELECT AVG(total)
    -> FROM (SELECT SUM(GoalsScored) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)
    -> AND TotalAssists > (SELECT AVG(total)
    -> FROM (SELECT SUM(Assists) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub2;
```



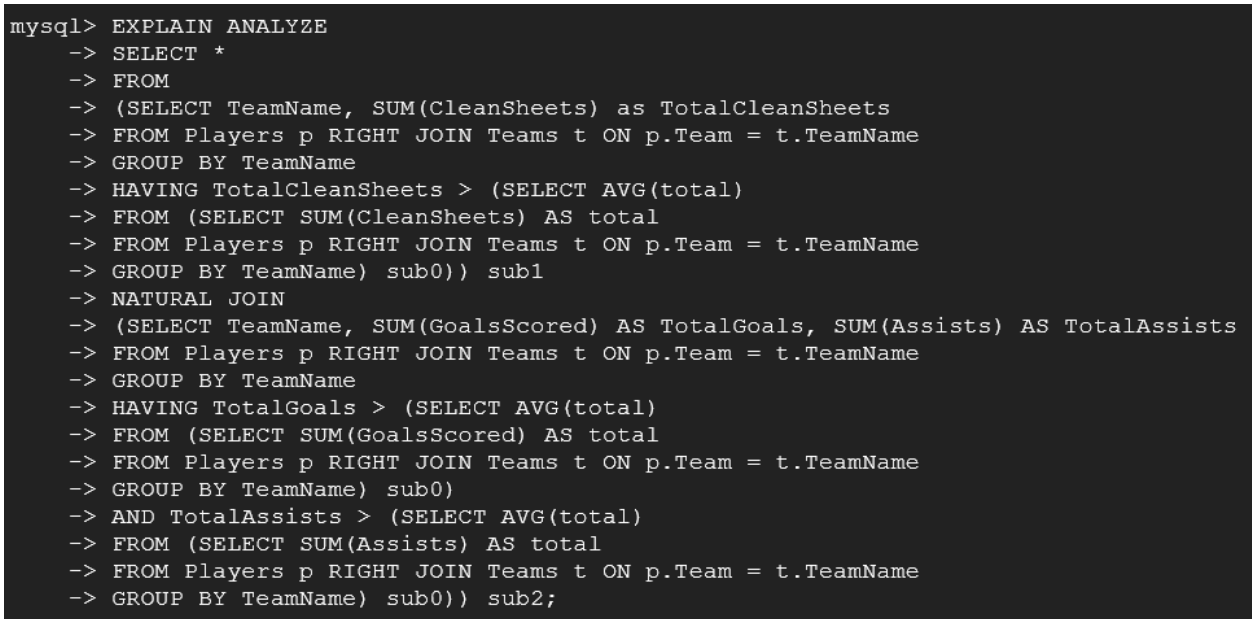
```
| -> Nested loop inner join  (cost=88007.50 rows=0) (actual time=13.274..13.286 rows=7 loops=1)
    -> Table scan on sub1  (cost=2.50..2.50 rows=0) (actual time=4.632..4.634 rows=9 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=4.631..4.631 rows=9 loops=1)
            -> Filter: (TotalCleanSheets > (select #3))  (actual time=4.606..4.614 rows=9 loops=1)
                -> Table scan on <temporary>  (actual time=2.266..2.273 rows=20 loops=1)
                    -> Aggregate using temporary table  (actual time=2.265..2.265 rows=20 loops=1)
                        -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.229..1.342 rows=1312 loops=1)
                            -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.018..0.023 rows=20 loops=1)
                            -> Hash
                                -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.079..0.609 rows=1676 loops=1)
                -> Select #3 (subquery in condition; run only once)
                    -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.234..2.234 rows=1 loops=1)
                        -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.225..2.227 rows=20 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.225..2.225 rows=20 loops=1)
                                -> Table scan on <temporary>  (actual time=2.213..2.216 rows=20 loops=1)
                                    -> Aggregate using temporary table  (actual time=2.213..2.213 rows=20 loops=1)
                                        -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.215..1.327 rows=1312 loops=1)
                                            -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.011..0.015 rows=20 loops=1)
                                            -> Hash
                                                -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.043..0.559 rows=1676 loops=1)
    -> Index lookup on sub2 using <auto_key0> (TeamName=sub1.TeamName)  (actual time=0.961..0.961 rows=1 loops=9)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=8.635..8.635 rows=9 loops=1)
            -> Filter: ((TotalGoals > (select #6)) and (TotalAssists > (select #8)))  (actual time=8.604..8.618 rows=9 loops=1)
                -> Table scan on <temporary>  (actual time=2.399..2.407 rows=20 loops=1)
                    -> Aggregate using temporary table  (actual time=2.398..2.398 rows=20 loops=1)
                        -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.246..1.364 rows=1312 loops=1)
                            -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.011..0.014 rows=20 loops=1)
                            -> Hash
                                -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.042..0.564 rows=1676 loops=1)
```



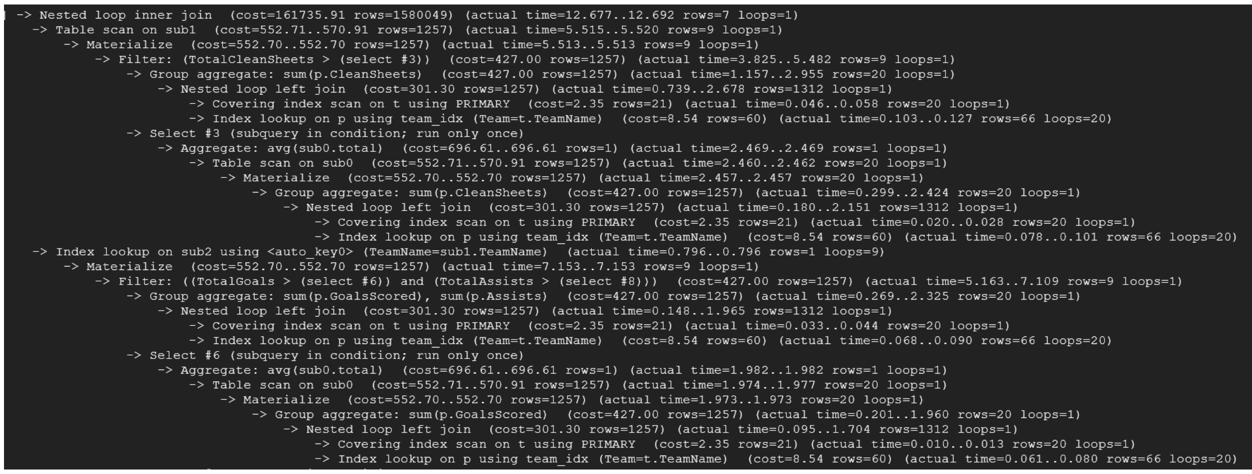
```
                -> Select #6 (subquery in condition; run only once)
                    -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.948..2.948 rows=1 loops=1)
                        -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.939..2.941 rows=20 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.938..2.938 rows=20 loops=1)
                                -> Table scan on <temporary>  (actual time=2.920..2.923 rows=20 loops=1)
                                    -> Aggregate using temporary table  (actual time=2.919..2.919 rows=20 loops=1)
                                        -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.817..1.961 rows=1312 loops=1)
                                            -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.079..0.086 rows=20 loops=1)
                                            -> Hash
                                                -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.442..1.054 rows=1676 loops=1)
                -> Select #8 (subquery in condition; run only once)
                    -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.275..2.275 rows=1 loops=1)
                        -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.267..2.269 rows=20 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.267..2.267 rows=20 loops=1)
                                -> Table scan on <temporary>  (actual time=2.253..2.255 rows=20 loops=1)
                                    -> Aggregate using temporary table  (actual time=2.252..2.252 rows=20 loops=1)
                                        -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.189..1.317 rows=1312 loops=1)
                                            -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.010..0.015 rows=20 loops=1)
                                            -> Hash
                                                -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.044..0.584 rows=1676 loops=1)
```

1. Indexing on Players(Team)

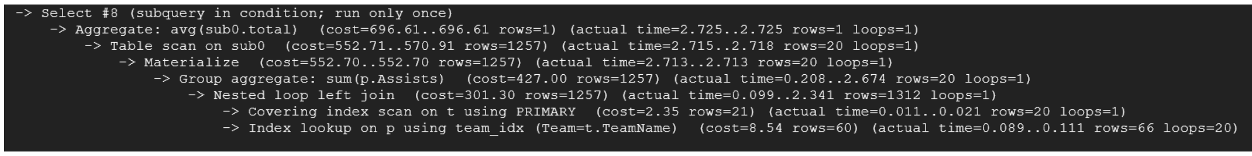
a.



```
mysql> EXPLAIN ANALYZE
    -> SELECT *
    -> FROM
    -> (SELECT TeamName, SUM(CleanSheets) as TotalCleanSheets
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalCleanSheets > (SELECT AVG(total)
    -> FROM (SELECT SUM(CleanSheets) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub1
    -> NATURAL JOIN
    -> (SELECT TeamName, SUM(GoalsScored) AS TotalGoals, SUM(Assists) AS TotalAssists
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalGoals > (SELECT AVG(total)
    -> FROM (SELECT SUM(GoalsScored) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)
    -> AND TotalAssists > (SELECT AVG(total)
    -> FROM (SELECT SUM(Assists) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub2;
```



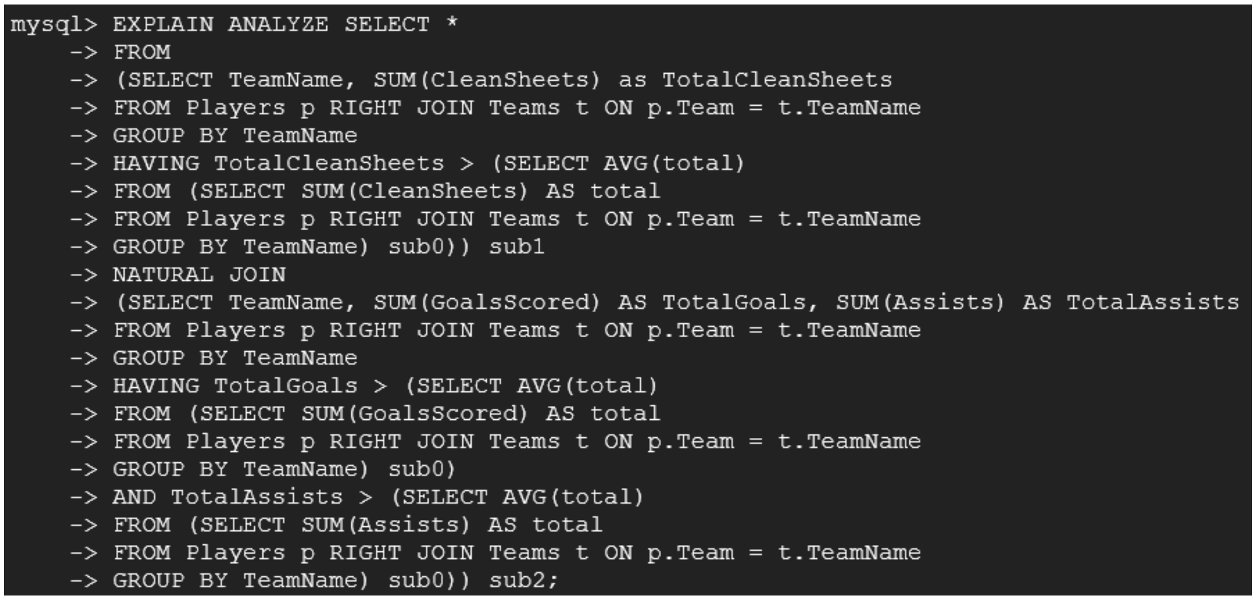
```
-> Nested loop inner join  (cost=161735.91 rows=1580049) (actual time=12.677..12.692 rows=7 loops=1)
    -> Table scan on sub1  (cost=552.71..570.91 rows=1257) (actual time=5.515..5.520 rows=9 loops=1)
        -> Materialize  (cost=552.70..552.70 rows=1257) (actual time=5.513..5.513 rows=9 loops=1)
            -> Filter: (TotalCleanSheets > (select #3))  (cost=427.00 rows=1257) (actual time=3.825..5.482 rows=9 loops=1)
                -> Group aggregate: sum(p.CleanSheets)  (cost=427.00 rows=1257) (actual time=1.157..2.955 rows=20 loops=1)
                    -> Nested loop left join  (cost=301.30 rows=1257) (actual time=0.739..2.678 rows=1312 loops=1)
                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.046..0.058 rows=20 loops=1)
                        -> Index lookup on p using team_idx (Team=t.TeamName)  (cost=8.54 rows=60) (actual time=0.103..0.127 rows=66 loops=20)
                -> Select #3 (subquery in condition; run only once)
                    -> Aggregate: avg(sub0.total)  (cost=696.61..696.61 rows=1) (actual time=2.469..2.469 rows=1 loops=1)
                        -> Table scan on sub0  (cost=552.71..570.91 rows=1257) (actual time=2.460..2.462 rows=20 loops=1)
                            -> Materialize  (cost=552.70..552.70 rows=1257) (actual time=2.457..2.457 rows=20 loops=1)
                                -> Group aggregate: sum(p.CleanSheets)  (cost=427.00 rows=1257) (actual time=0.299..2.424 rows=20 loops=1)
                                    -> Nested loop left join  (cost=301.30 rows=1257) (actual time=0.180..2.151 rows=1312 loops=1)
                                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.020..0.028 rows=20 loops=1)
                                        -> Index lookup on p using team_idx (Team=t.TeamName)  (cost=8.54 rows=60) (actual time=0.078..0.101 rows=66 loops=20)
    -> Index lookup on sub2 using <auto_key0> (TeamName=sub1.TeamName)  (actual time=0.796..0.796 rows=1 loops=9)
        -> Materialize  (cost=552.70..552.70 rows=1257) (actual time=7.153..7.153 rows=9 loops=1)
            -> Filter: ((TotalGoals > (select #6)) and (TotalAssists > (select #8)))  (cost=427.00 rows=1257) (actual time=5.163..7.109 rows=9 loops=1)
                -> Group aggregate: sum(p.GoalsScored), sum(p.Assists)  (cost=427.00 rows=1257) (actual time=0.269..2.325 rows=20 loops=1)
                    -> Nested loop left join  (cost=301.30 rows=1257) (actual time=0.148..1.965 rows=1312 loops=1)
                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.033..0.044 rows=20 loops=1)
                        -> Index lookup on p using team_idx (Team=t.TeamName)  (cost=8.54 rows=60) (actual time=0.068..0.090 rows=66 loops=20)
                -> Select #6 (subquery in condition; run only once)
                    -> Aggregate: avg(sub0.total)  (cost=696.61..696.61 rows=1) (actual time=1.982..1.982 rows=1 loops=1)
                        -> Table scan on sub0  (cost=552.71..570.91 rows=1257) (actual time=1.974..1.977 rows=20 loops=1)
                            -> Materialize  (cost=552.70..552.70 rows=1257) (actual time=1.973..1.973 rows=20 loops=1)
                                -> Group aggregate: sum(p.GoalsScored)  (cost=427.00 rows=1257) (actual time=0.201..1.960 rows=20 loops=1)
                                    -> Nested loop left join  (cost=301.30 rows=1257) (actual time=0.095..1.704 rows=1312 loops=1)
                                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.010..0.013 rows=20 loops=1)
                                        -> Index lookup on p using team_idx (Team=t.TeamName)  (cost=8.54 rows=60) (actual time=0.061..0.080 rows=66 loops=20)
```



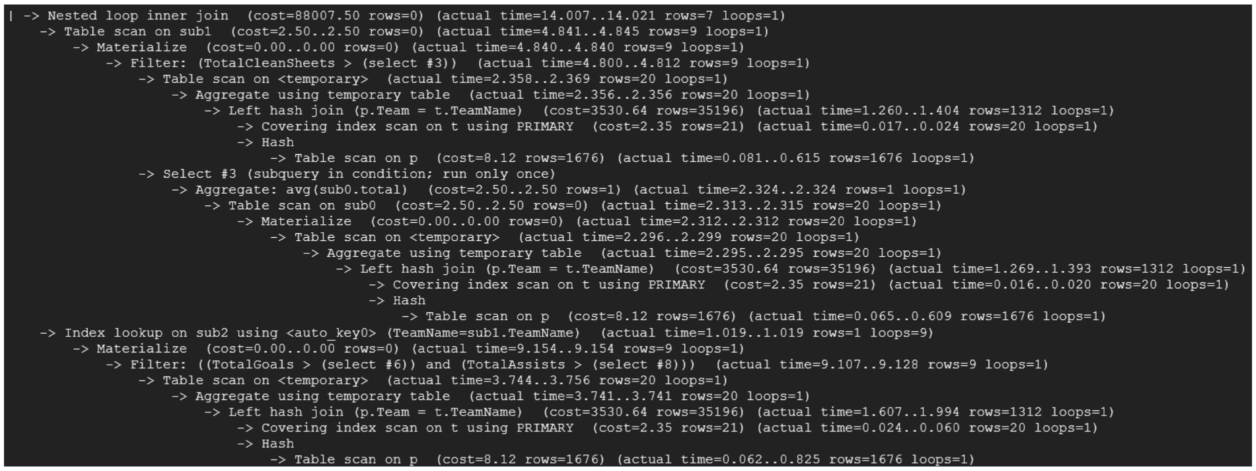
```
-> Select #8 (subquery in condition; run only once)
    -> Aggregate: avg(sub0.total)  (cost=696.61..696.61 rows=1) (actual time=2.725..2.725 rows=1 loops=1)
        -> Table scan on sub0  (cost=552.71..570.91 rows=1257) (actual time=2.715..2.718 rows=20 loops=1)
            -> Materialize  (cost=552.70..552.70 rows=1257) (actual time=2.713..2.713 rows=20 loops=1)
                -> Group aggregate: sum(p.Assists)  (cost=427.00 rows=1257) (actual time=0.208..2.674 rows=20 loops=1)
                    -> Nested loop left join  (cost=301.30 rows=1257) (actual time=0.099..2.341 rows=1312 loops=1)
                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.011..0.021 rows=20 loops=1)
                        -> Index lookup on p using team_idx (Team=t.TeamName)  (cost=8.54 rows=60) (actual time=0.089..0.111 rows=66 loops=20)
```

b. This almost doubled the cost of the query. I believe this is because of the low amount of rows we needed to actually join. Research reveals that it is possible for the query processor to spend more time deciding whether or not to use an index in some rare cases, so perhaps that is why. Also we have a lot many players as compared to the teams which is trivial in number and probably that is just making the case worse for indexing.

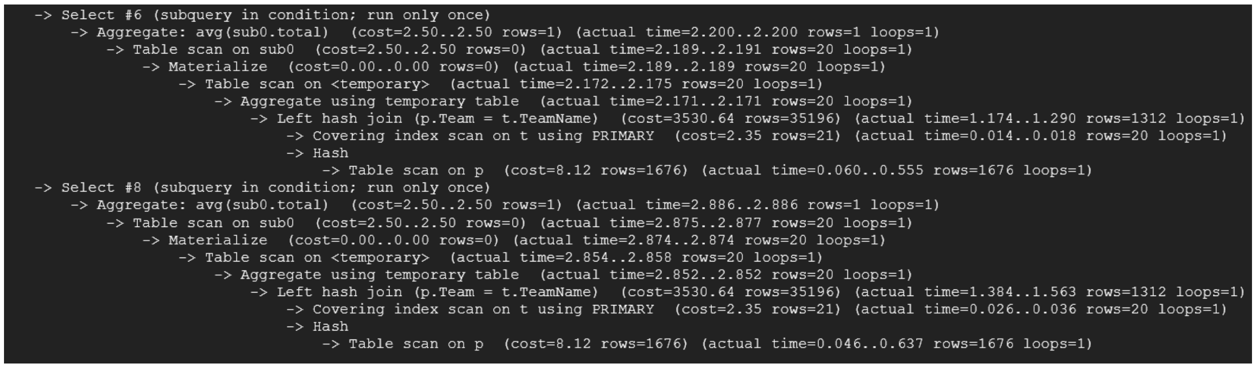
2. Indexing on Players(CleanSheets)



```
mysql> EXPLAIN ANALYZE SELECT *
    -> FROM
    -> (SELECT TeamName, SUM(CleanSheets) as TotalCleanSheets
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalCleanSheets > (SELECT AVG(total)
    -> FROM (SELECT SUM(CleanSheets) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub1
    -> NATURAL JOIN
    -> (SELECT TeamName, SUM(GoalsScored) AS TotalGoals, SUM(Assists) AS TotalAssists
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalGoals > (SELECT AVG(total)
    -> FROM (SELECT SUM(GoalsScored) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)
    -> AND TotalAssists > (SELECT AVG(total)
    -> FROM (SELECT SUM(Assists) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub2;
```



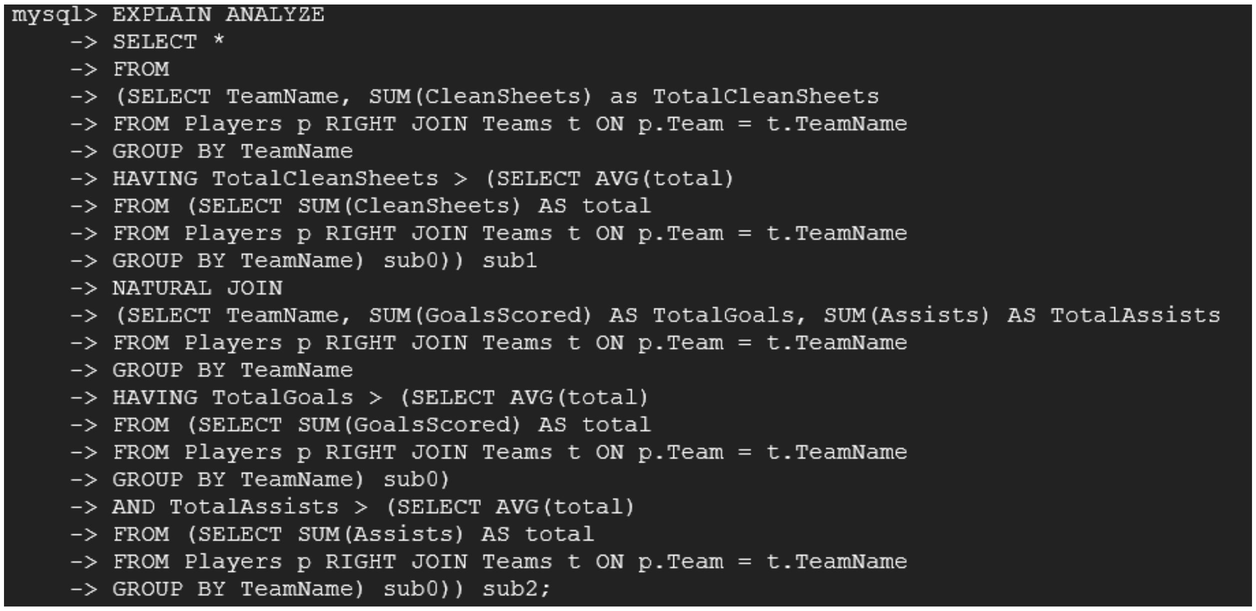
```
| -> Nested loop inner join  (cost=88007.50 rows=0) (actual time=14.007..14.021 rows=7 loops=1)
    -> Table scan on sub1  (cost=2.50..2.50 rows=0) (actual time=4.841..4.845 rows=9 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=4.840..4.840 rows=9 loops=1)
            -> Filter: (TotalCleanSheets > (select #3))  (actual time=4.800..4.812 rows=9 loops=1)
                -> Table scan on <temporary>  (actual time=2.358..2.369 rows=20 loops=1)
                    -> Aggregate using temporary table  (actual time=2.356..2.356 rows=20 loops=1)
                        -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.260..1.404 rows=1312 loops=1)
                            -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.017..0.024 rows=20 loops=1)
                            -> Hash
                                -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.081..0.615 rows=1676 loops=1)
                -> Select #3 (subquery in condition; run only once)
                    -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.324..2.324 rows=1 loops=1)
                        -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.313..2.315 rows=20 loops=1)
                            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.312..2.312 rows=20 loops=1)
                                -> Table scan on <temporary>  (actual time=2.296..2.299 rows=20 loops=1)
                                    -> Aggregate using temporary table  (actual time=2.295..2.295 rows=20 loops=1)
                                        -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.269..1.393 rows=1312 loops=1)
                                            -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.016..0.020 rows=20 loops=1)
                                            -> Hash
                                                -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.065..0.609 rows=1676 loops=1)
    -> Index lookup on sub2 using <auto_key0> (TeamName=sub1.TeamName)  (actual time=1.019..1.019 rows=1 loops=9)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=9.154..9.154 rows=9 loops=1)
            -> Filter: ((TotalGoals > (select #6)) and (TotalAssists > (select #8)))  (actual time=9.107..9.128 rows=9 loops=1)
                -> Table scan on <temporary>  (actual time=3.744..3.756 rows=20 loops=1)
                    -> Aggregate using temporary table  (actual time=3.741..3.741 rows=20 loops=1)
                        -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.607..1.994 rows=1312 loops=1)
                            -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.024..0.060 rows=20 loops=1)
                            -> Hash
                                -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.062..0.825 rows=1676 loops=1)
```



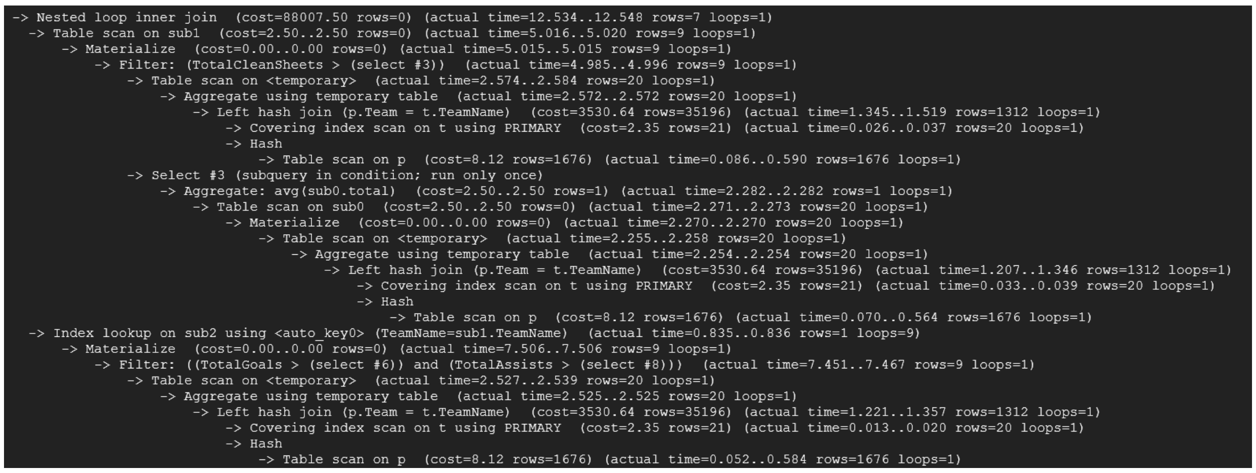
```
   -> Select #6 (subquery in condition; run only once)
       -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.200..2.200 rows=1 loops=1)
           -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.189..2.191 rows=20 loops=1)
               -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.189..2.189 rows=20 loops=1)
                   -> Table scan on <temporary>  (actual time=2.172..2.175 rows=20 loops=1)
                       -> Aggregate using temporary table  (actual time=2.171..2.171 rows=20 loops=1)
                           -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.174..1.290 rows=1312 loops=1)
                               -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.014..0.018 rows=20 loops=1)
                               -> Hash
                                   -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.060..0.555 rows=1676 loops=1)
   -> Select #8 (subquery in condition; run only once)
       -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.886..2.886 rows=1 loops=1)
           -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.875..2.877 rows=20 loops=1)
               -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.874..2.874 rows=20 loops=1)
                   -> Table scan on <temporary>  (actual time=2.854..2.858 rows=20 loops=1)
                       -> Aggregate using temporary table  (actual time=2.852..2.852 rows=20 loops=1)
                           -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.384..1.563 rows=1312 loops=1)
                               -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.026..0.036 rows=20 loops=1)
                               -> Hash
                                   -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.046..0.637 rows=1676 loops=1)
```

a. Same performance as without indexing. It likely did not help as we used the CleanSheets attribute in SUM() and AVG() aggregations, which do not benefit from indexing since those operations have to scan the entire attribute anyway.Moreover there could be similar values for the CleanSheets for players that is not helping the indexing .We also have a group by in place for the teams and maybe that is not adding much benefit if we index based on cleansheets.

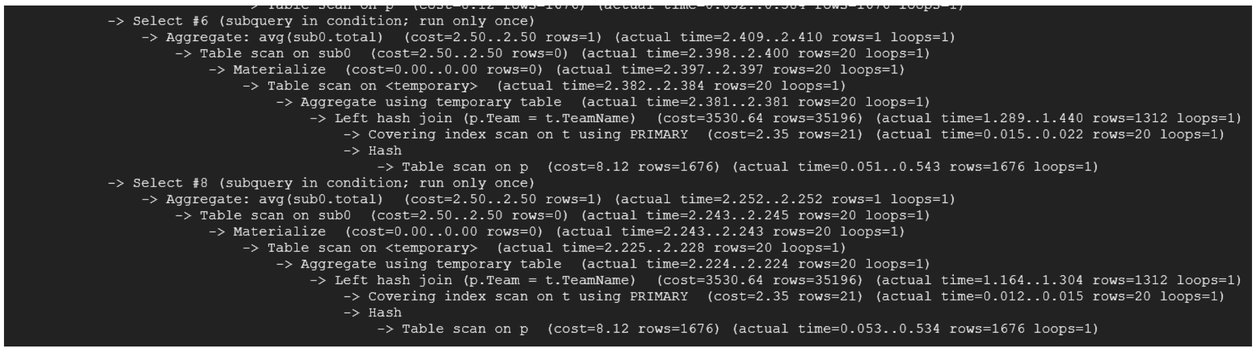
3. Indexing on Players(GoalsScored, Assists)



```
mysql> EXPLAIN ANALYZE
    -> SELECT *
    -> FROM
    -> (SELECT TeamName, SUM(CleanSheets) as TotalCleanSheets
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalCleanSheets > (SELECT AVG(total)
    -> FROM (SELECT SUM(CleanSheets) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub1
    -> NATURAL JOIN
    -> (SELECT TeamName, SUM(GoalsScored) AS TotalGoals, SUM(Assists) AS TotalAssists
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName
    -> HAVING TotalGoals > (SELECT AVG(total)
    -> FROM (SELECT SUM(GoalsScored) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)
    -> AND TotalAssists > (SELECT AVG(total)
    -> FROM (SELECT SUM(Assists) AS total
    -> FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    -> GROUP BY TeamName) sub0)) sub2;
```



```
-> Nested loop inner join  (cost=88007.50 rows=0) (actual time=12.534..12.548 rows=7 loops=1)
  -> Table scan on sub1  (cost=2.50..2.50 rows=0) (actual time=5.016..5.020 rows=9 loops=1)
      -> Materialize  (cost=0.00..0.00 rows=0) (actual time=5.015..5.015 rows=9 loops=1)
          -> Filter: (TotalCleanSheets > (select #3))  (actual time=4.985..4.996 rows=9 loops=1)
              -> Table scan on <temporary>  (actual time=2.574..2.584 rows=20 loops=1)
                  -> Aggregate using temporary table  (actual time=2.572..2.572 rows=20 loops=1)
                      -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.345..1.519 rows=1312 loops=1)
                          -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.026..0.037 rows=20 loops=1)
                          -> Hash
                              -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.086..0.590 rows=1676 loops=1)
              -> Select #3 (subquery in condition; run only once)
                  -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.282..2.282 rows=1 loops=1)
                      -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.271..2.273 rows=20 loops=1)
                          -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.270..2.270 rows=20 loops=1)
                              -> Table scan on <temporary>  (actual time=2.255..2.258 rows=20 loops=1)
                                  -> Aggregate using temporary table  (actual time=2.254..2.254 rows=20 loops=1)
                                      -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.207..1.346 rows=1312 loops=1)
                                          -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.033..0.039 rows=20 loops=1)
                                          -> Hash
                                              -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.070..0.564 rows=1676 loops=1)
  -> Index lookup on sub2 using <auto_key0> (TeamName=sub1.TeamName)  (actual time=0.835..0.836 rows=1 loops=9)
      -> Materialize  (cost=0.00..0.00 rows=0) (actual time=7.506..7.506 rows=9 loops=1)
          -> Filter: ((TotalGoals > (select #6)) and (TotalAssists > (select #8)))  (actual time=7.451..7.467 rows=9 loops=1)
              -> Table scan on <temporary>  (actual time=2.527..2.539 rows=20 loops=1)
                  -> Aggregate using temporary table  (actual time=2.525..2.525 rows=20 loops=1)
                      -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.221..1.357 rows=1312 loops=1)
                          -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.013..0.020 rows=20 loops=1)
                          -> Hash
                              -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.052..0.584 rows=1676 loops=1)
```



```
            -> Select #6 (subquery in condition; run only once)
                -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.409..2.410 rows=1 loops=1)
                    -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.398..2.400 rows=20 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.397..2.397 rows=20 loops=1)
                            -> Table scan on <temporary>  (actual time=2.382..2.384 rows=20 loops=1)
                                -> Aggregate using temporary table  (actual time=2.381..2.381 rows=20 loops=1)
                                    -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.289..1.440 rows=1312 loops=1)
                                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.015..0.022 rows=20 loops=1)
                                        -> Hash
                                            -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.051..0.543 rows=1676 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Aggregate: avg(sub0.total)  (cost=2.50..2.50 rows=1) (actual time=2.252..2.252 rows=1 loops=1)
                    -> Table scan on sub0  (cost=2.50..2.50 rows=0) (actual time=2.243..2.245 rows=20 loops=1)
                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.243..2.243 rows=20 loops=1)
                            -> Table scan on <temporary>  (actual time=2.225..2.228 rows=20 loops=1)
                                -> Aggregate using temporary table  (actual time=2.224..2.224 rows=20 loops=1)
                                    -> Left hash join (p.Team = t.TeamName)  (cost=3530.64 rows=35196) (actual time=1.164..1.304 rows=1312 loops=1)
                                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.012..0.015 rows=20 loops=1)
                                        -> Hash
                                            -> Table scan on p  (cost=8.12 rows=1676) (actual time=0.053..0.534 rows=1676 loops=1)
```

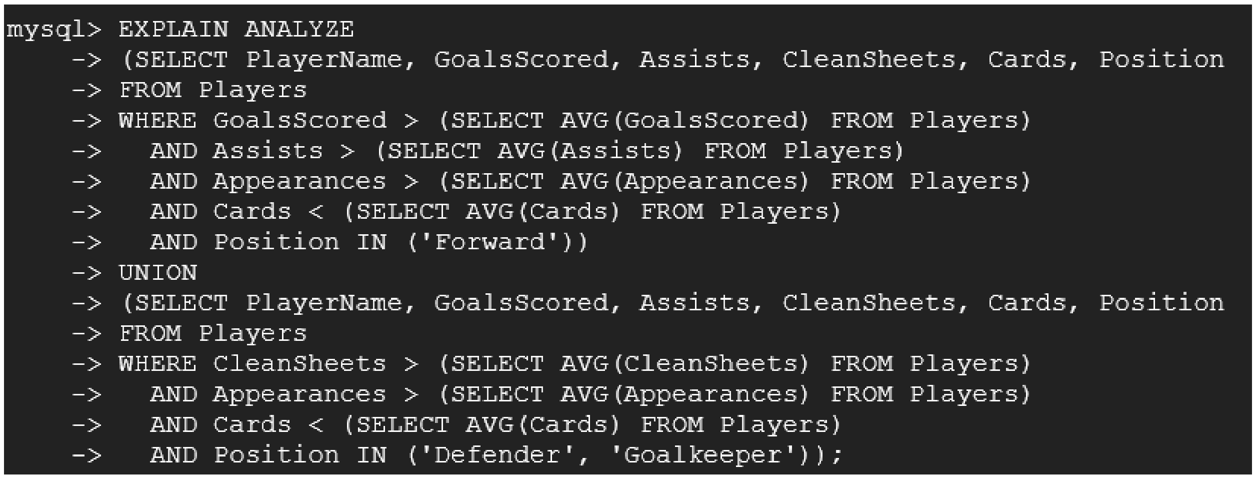
   a. No performance improvement, likely for the same reason as 2. Because we use these attributes in aggregation queries. Same performance as without indexing. It likely did not help as we used the (GoalsScored, Assists) attribute in SUM() and AVG() aggregations, which do not benefit from indexing since those operations have to scan the entire attribute anyway.Moreover there could be similar values for the (GoalsScored, Assists) for players that is not helping the indexing .We also have a group by in place for the teams and maybe that is not adding much benefit if we index based on (GoalsScored, Assists).

b.

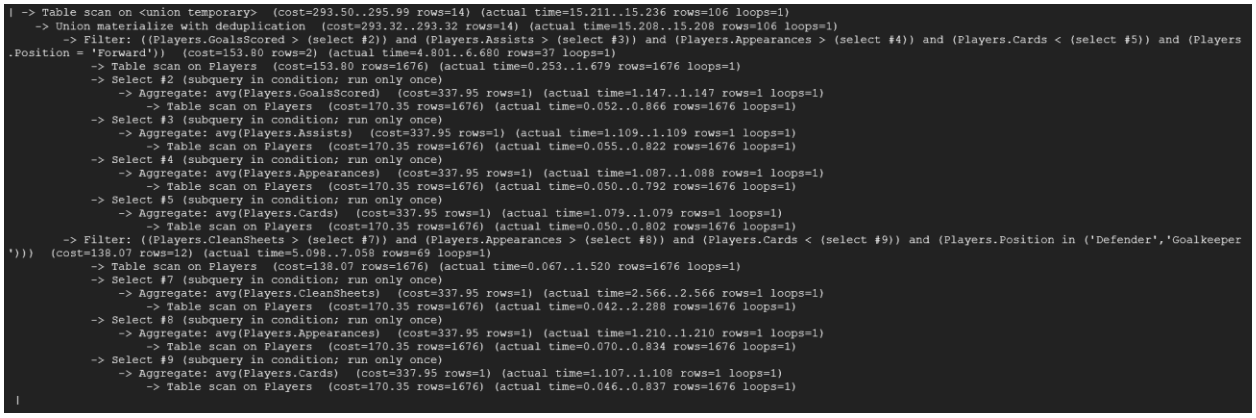
4. **FINAL SCHEME:** No indexing
   We chose not to index this query because we could not yield any improvements.

Query 2

Before Indexing

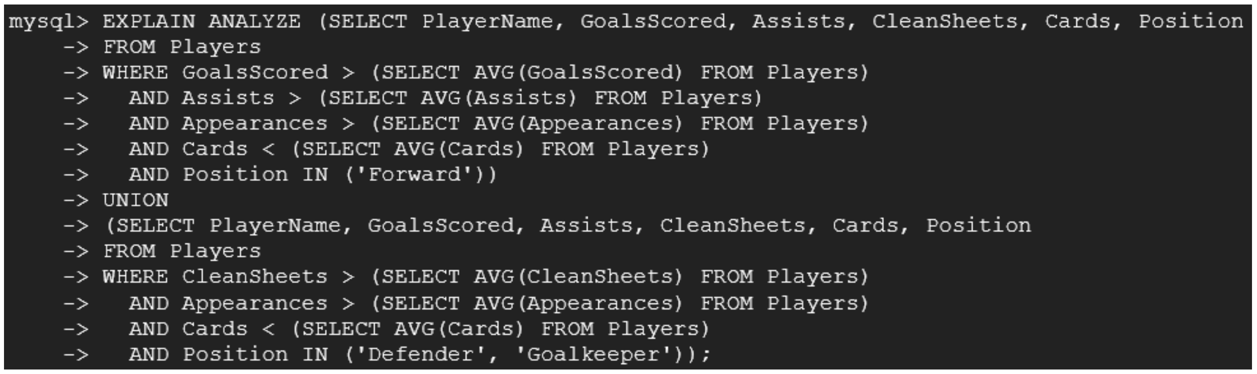


```
mysql> EXPLAIN ANALYZE
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   AND Assists > (SELECT AVG(Assists) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Forward'))
    -> UNION
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Defender', 'Goalkeeper'));
```

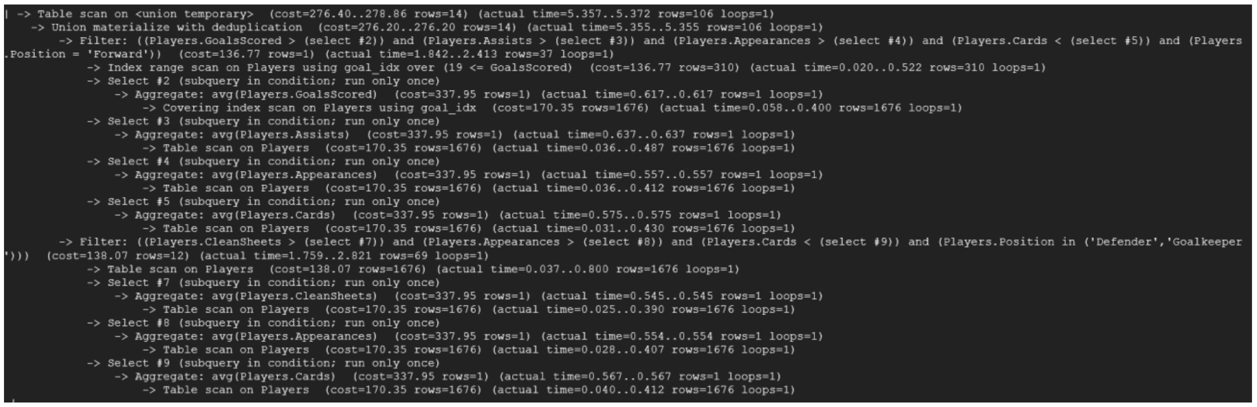


```
| -> Table scan on <union temporary>  (cost=293.50..295.99 rows=14) (actual time=15.211..15.236 rows=106 loops=1)
    -> Union materialize with deduplication  (cost=293.32..293.32 rows=14) (actual time=15.208..15.208 rows=106 loops=1)
        -> Filter: ((Players.GoalsScored > (select #2)) and (Players.Assists > (select #3)) and (Players.Appearances > (select #4)) and (Players.Cards < (select #5)) and (Players
.Position = 'Forward'))  (cost=153.80 rows=2) (actual time=4.801..6.680 rows=37 loops=1)
            -> Table scan on Players  (cost=153.80 rows=1676) (actual time=0.253..1.679 rows=1676 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=1.147..1.147 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.052..0.866 rows=1676 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=1.109..1.109 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.055..0.822 rows=1676 loops=1)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=1.087..1.088 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.050..0.792 rows=1676 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=1.079..1.079 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.050..0.802 rows=1676 loops=1)
        -> Filter: ((Players.CleanSheets > (select #7)) and (Players.Appearances > (select #8)) and (Players.Cards < (select #9)) and (Players.Position in ('Defender','Goalkeeper
')))  (cost=138.07 rows=12) (actual time=5.098..7.058 rows=69 loops=1)
            -> Table scan on Players  (cost=138.07 rows=1676) (actual time=0.067..1.520 rows=1676 loops=1)
            -> Select #7 (subquery in condition; run only once)
                -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=2.566..2.566 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.042..2.288 rows=1676 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=1.210..1.210 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.070..0.834 rows=1676 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=1.107..1.108 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.046..0.837 rows=1676 loops=1)
 |
```

1. Indexing on Players(GoalsScored)



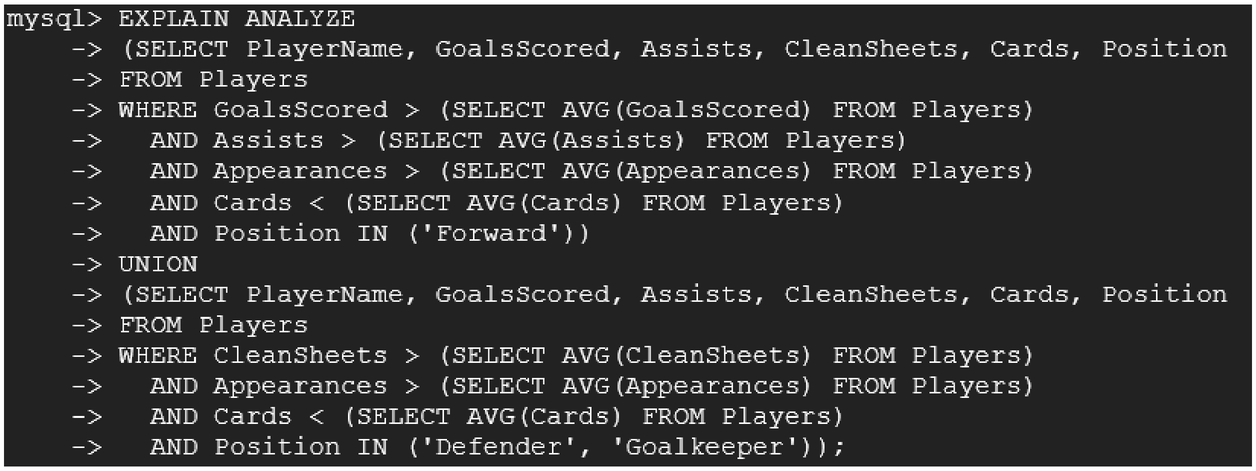
```
mysql> EXPLAIN ANALYZE (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   AND Assists > (SELECT AVG(Assists) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Forward'))
    -> UNION
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Defender', 'Goalkeeper'));
```



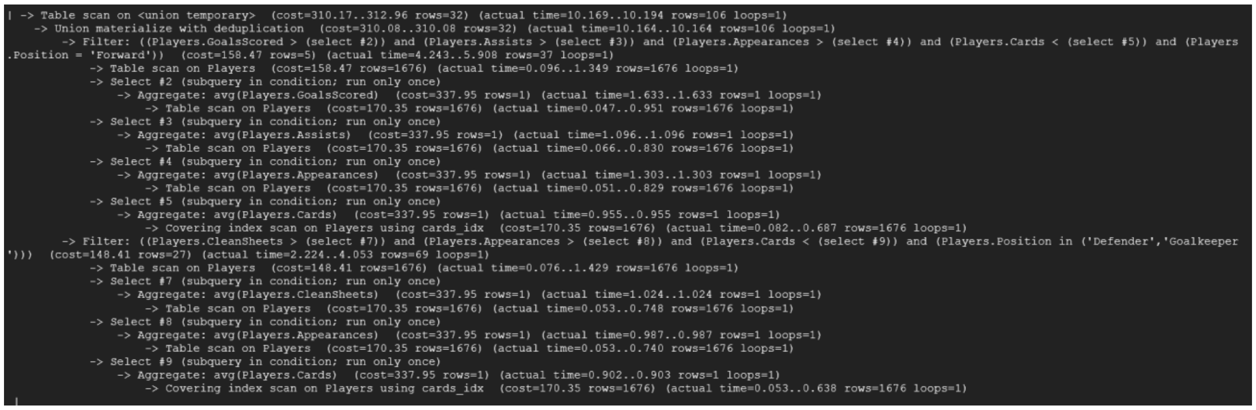
```
| -> Table scan on <union temporary>  (cost=276.40..278.86 rows=14) (actual time=5.357..5.372 rows=106 loops=1)
    -> Union materialize with deduplication  (cost=276.20..276.20 rows=14) (actual time=5.355..5.355 rows=106 loops=1)
        -> Filter: ((Players.GoalsScored > (select #2)) and (Players.Assists > (select #3)) and (Players.Appearances > (select #4)) and (Players.Cards < (select #5)) and (Players
.Position = 'Forward'))  (cost=136.77 rows=1) (actual time=1.842..2.413 rows=37 loops=1)
            -> Index range scan on Players using goal_idx over (19 <= GoalsScored)  (cost=136.77 rows=310) (actual time=0.020..0.522 rows=310 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=0.617..0.617 rows=1 loops=1)
                    -> Covering index scan on Players using goal_idx  (cost=170.35 rows=1676) (actual time=0.058..0.400 rows=1676 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=0.637..0.637 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.036..0.487 rows=1676 loops=1)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=0.557..0.557 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.036..0.412 rows=1676 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=0.575..0.575 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.031..0.430 rows=1676 loops=1)
        -> Filter: ((Players.CleanSheets > (select #7)) and (Players.Appearances > (select #8)) and (Players.Cards < (select #9)) and (Players.Position in ('Defender','Goalkeeper
')))  (cost=138.07 rows=12) (actual time=1.759..2.821 rows=69 loops=1)
            -> Table scan on Players  (cost=138.07 rows=1676) (actual time=0.037..0.800 rows=1676 loops=1)
            -> Select #7 (subquery in condition; run only once)
                -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=0.545..0.545 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.025..0.390 rows=1676 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=0.554..0.554 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.028..0.407 rows=1676 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=0.567..0.567 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.040..0.412 rows=1676 loops=1)
```

   a. Small performance increases. We compare GoalsScored to AVG(GoalsScored) in a WHERE clause which is likely why. We use GoalsScored outside of an aggregation operation here.

2. Indexing on Players(Cards)



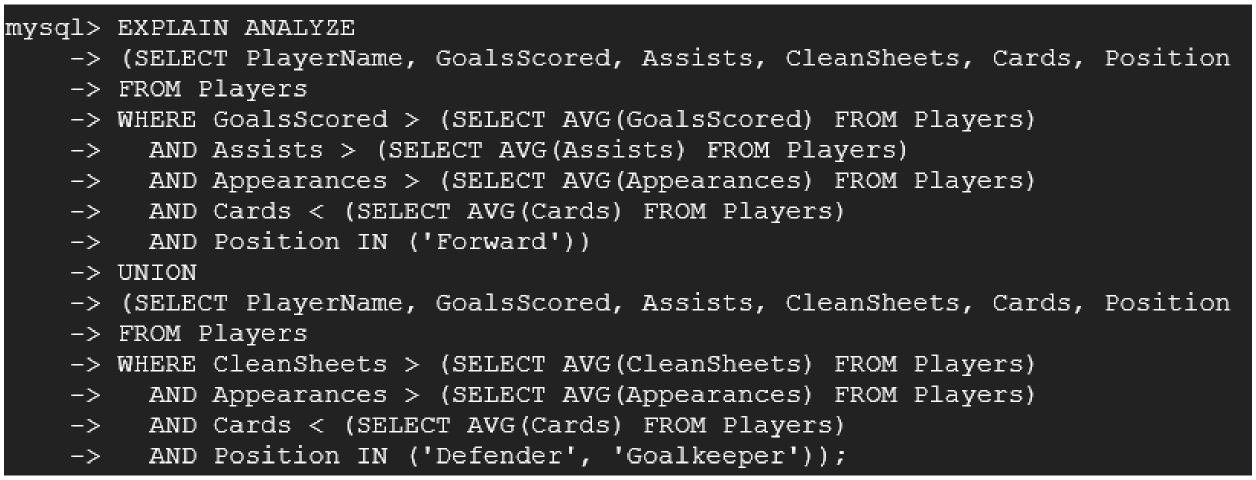
```
mysql> EXPLAIN ANALYZE
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   AND Assists > (SELECT AVG(Assists) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Forward'))
    -> UNION
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Defender', 'Goalkeeper'));
```



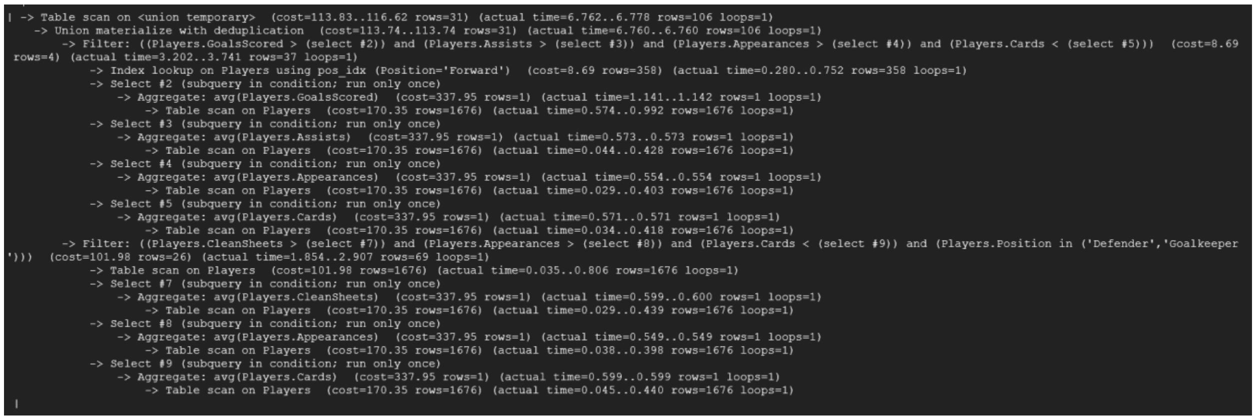
```
| -> Table scan on <union temporary>  (cost=310.17..312.96 rows=32) (actual time=10.169..10.194 rows=106 loops=1)
    -> Union materialize with deduplication  (cost=310.08..310.08 rows=32) (actual time=10.164..10.164 rows=106 loops=1)
        -> Filter: ((Players.GoalsScored > (select #2)) and (Players.Assists > (select #3)) and (Players.Appearances > (select #4)) and (Players.Cards < (select #5)) and (Players
.Position = 'Forward'))  (cost=158.47 rows=5) (actual time=4.243..5.908 rows=37 loops=1)
            -> Table scan on Players  (cost=158.47 rows=1676) (actual time=0.096..1.349 rows=1676 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=1.633..1.633 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.047..0.951 rows=1676 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=1.096..1.096 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.066..0.830 rows=1676 loops=1)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=1.303..1.303 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.051..0.829 rows=1676 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=0.955..0.955 rows=1 loops=1)
                    -> Covering index scan on Players using cards_idx  (cost=170.35 rows=1676) (actual time=0.082..0.687 rows=1676 loops=1)
        -> Filter: ((Players.CleanSheets > (select #7)) and (Players.Appearances > (select #8)) and (Players.Cards < (select #9)) and (Players.Position in ('Defender','Goalkeeper
')))  (cost=148.41 rows=27) (actual time=2.224..4.053 rows=69 loops=1)
            -> Table scan on Players  (cost=148.41 rows=1676) (actual time=0.076..1.429 rows=1676 loops=1)
            -> Select #7 (subquery in condition; run only once)
                -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=1.024..1.024 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.053..0.748 rows=1676 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=0.987..0.987 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.053..0.740 rows=1676 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=0.902..0.903 rows=1 loops=1)
                    -> Covering index scan on Players using cards_idx  (cost=170.35 rows=1676) (actual time=0.053..0.638 rows=1676 loops=1)
 |
```

   a. Slightly worse performance. Cards does not appear to be an attribute that filters a lot of rows out quickly. This is likely due to the position of Cards in the WHERE clause, following many other ANDs. As a result, it probably takes more time for the query processor to decide to use the index than what we filter out because of a lot of similar values maybe.

3. Indexing on Players(Position)



```
mysql> EXPLAIN ANALYZE
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   AND Assists > (SELECT AVG(Assists) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Forward'))
    -> UNION
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Defender', 'Goalkeeper'));
```

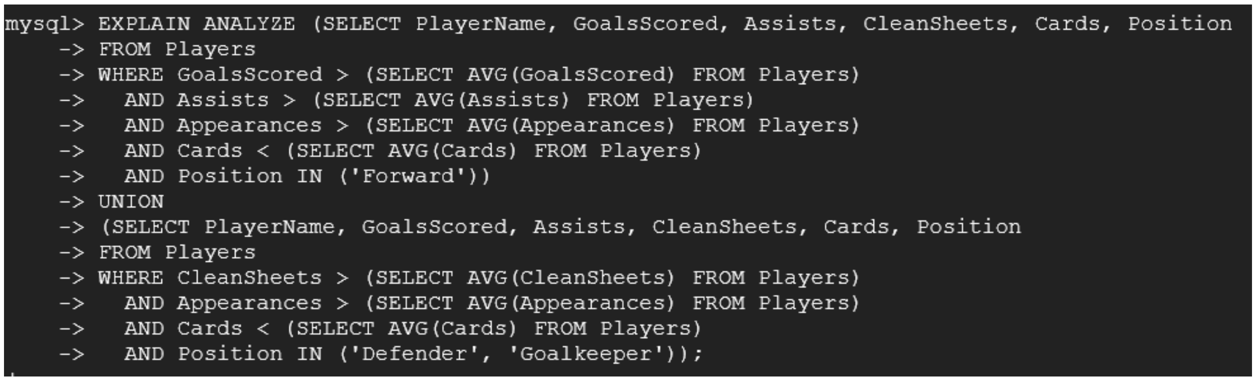


```
| -> Table scan on <union temporary>  (cost=113.83..116.62 rows=31) (actual time=6.762..6.778 rows=106 loops=1)
    -> Union materialize with deduplication  (cost=113.74..113.74 rows=31) (actual time=6.760..6.760 rows=106 loops=1)
        -> Filter: ((Players.GoalsScored > (select #2)) and (Players.Assists > (select #3)) and (Players.Appearances > (select #4)) and (Players.Cards < (select #5)))  (cost=8.69
 rows=4) (actual time=3.202..3.741 rows=37 loops=1)
            -> Index lookup on Players using pos_idx (Position='Forward')  (cost=8.69 rows=358) (actual time=0.280..0.752 rows=358 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=1.141..1.142 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.574..0.992 rows=1676 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=0.573..0.573 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.044..0.428 rows=1676 loops=1)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=0.554..0.554 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.029..0.403 rows=1676 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=0.571..0.571 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.034..0.418 rows=1676 loops=1)
        -> Filter: ((Players.CleanSheets > (select #7)) and (Players.Appearances > (select #8)) and (Players.Cards < (select #9)) and (Players.Position in ('Defender','Goalkeeper
')))  (cost=101.98 rows=26) (actual time=1.854..2.907 rows=69 loops=1)
            -> Table scan on Players  (cost=101.98 rows=1676) (actual time=0.035..0.806 rows=1676 loops=1)
            -> Select #7 (subquery in condition; run only once)
                -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=0.599..0.600 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.029..0.439 rows=1676 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=0.549..0.549 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.038..0.398 rows=1676 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=0.599..0.599 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.045..0.440 rows=1676 loops=1)
 |
```

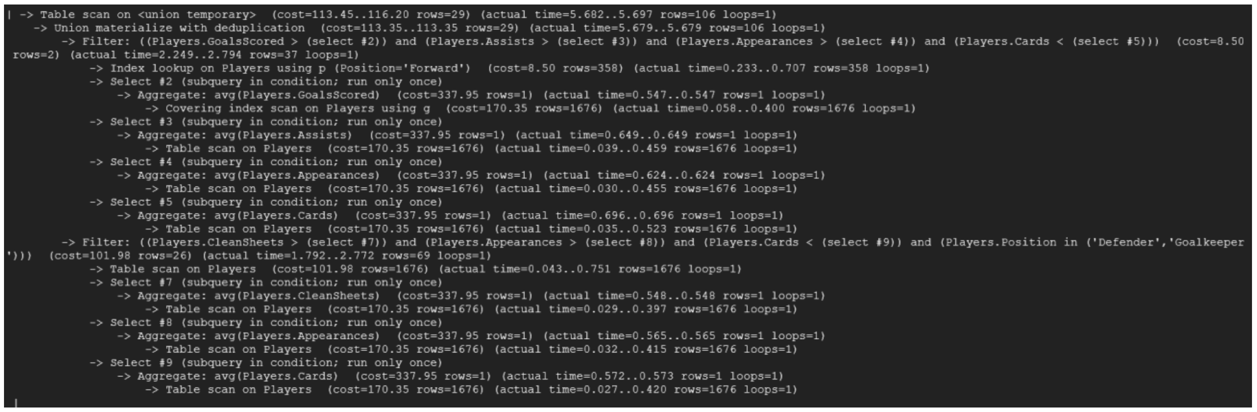
a. Most significant improvement so far, likely because an searching on VARCHAR() is more expensive than searching on INT and it directly helps us to filter out substantial amount of specific data (partition) because position is used in our where clause in both the cases of the union.

b.)

4. **FINAL SCHEME:** Indexing on Players(GoalsScored, Position)



```
mysql> EXPLAIN ANALYZE (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   AND Assists > (SELECT AVG(Assists) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Forward'))
    -> UNION
    -> (SELECT PlayerName, GoalsScored, Assists, CleanSheets, Cards, Position
    -> FROM Players
    -> WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players)
    ->   AND Appearances > (SELECT AVG(Appearances) FROM Players)
    ->   AND Cards < (SELECT AVG(Cards) FROM Players)
    ->   AND Position IN ('Defender', 'Goalkeeper'));
```

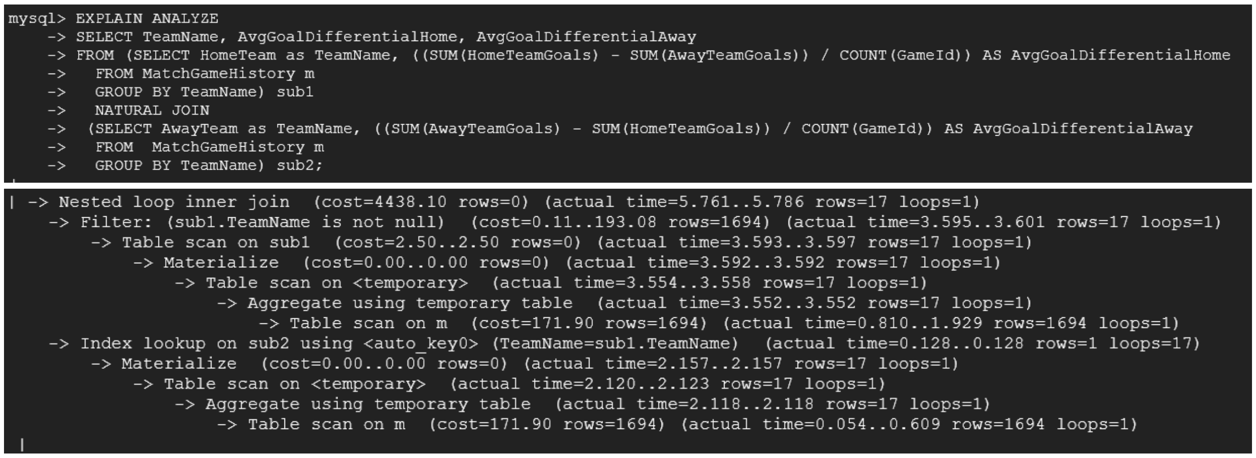


```
| -> Table scan on <union temporary>  (cost=113.45..116.20 rows=29) (actual time=5.682..5.697 rows=106 loops=1)
    -> Union materialize with deduplication  (cost=113.35..113.35 rows=29) (actual time=5.679..5.679 rows=106 loops=1)
        -> Filter: ((Players.GoalsScored > (select #2)) and (Players.Assists > (select #3)) and (Players.Appearances > (select #4)) and (Players.Cards < (select #5)))  (cost=8.50
 rows=2) (actual time=2.249..2.794 rows=37 loops=1)
            -> Index lookup on Players using p (Position='Forward')  (cost=8.50 rows=358) (actual time=0.233..0.707 rows=358 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=0.547..0.547 rows=1 loops=1)
                    -> Covering index scan on Players using g  (cost=170.35 rows=1676) (actual time=0.058..0.400 rows=1676 loops=1)
            -> Select #3 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=0.649..0.649 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.039..0.459 rows=1676 loops=1)
            -> Select #4 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=0.624..0.624 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.030..0.455 rows=1676 loops=1)
            -> Select #5 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=0.696..0.696 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.035..0.523 rows=1676 loops=1)
        -> Filter: ((Players.CleanSheets > (select #7)) and (Players.Appearances > (select #8)) and (Players.Cards < (select #9)) and (Players.Position in ('Defender','Goalkeeper
')))  (cost=101.98 rows=26) (actual time=1.792..2.772 rows=69 loops=1)
            -> Table scan on Players  (cost=101.98 rows=1676) (actual time=0.043..0.751 rows=1676 loops=1)
            -> Select #7 (subquery in condition; run only once)
                -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=0.548..0.548 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.029..0.397 rows=1676 loops=1)
            -> Select #8 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Appearances)  (cost=337.95 rows=1) (actual time=0.565..0.565 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.032..0.415 rows=1676 loops=1)
            -> Select #9 (subquery in condition; run only once)
                -> Aggregate: avg(Players.Cards)  (cost=337.95 rows=1) (actual time=0.572..0.573 rows=1 loops=1)
                    -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.027..0.420 rows=1676 loops=1)
 |
```

   a. We chose this combination of our three indexing schemes because it yielded the best performance. Arguably the increase from Position alone to Position including GoalsScored is not worth the additional space, but since our program is not occupying much space on GCP so far, it meets our technical specifications.

Query 3

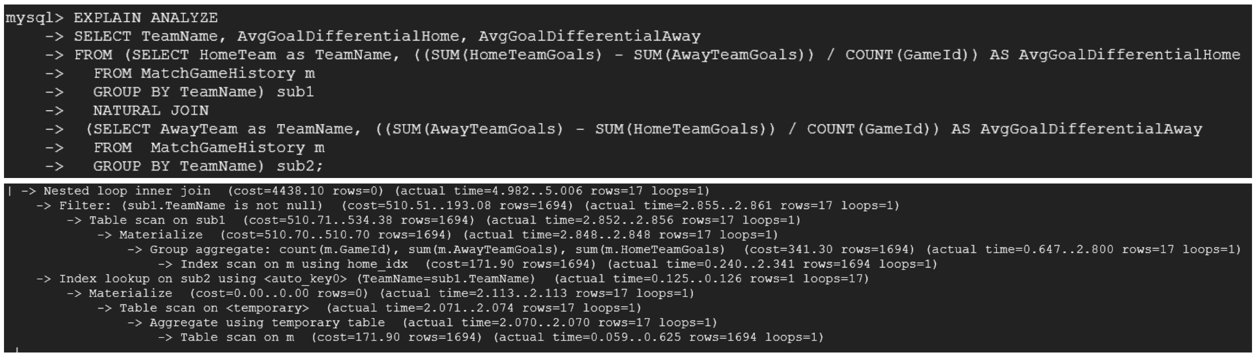
Before Indexing



```
mysql> EXPLAIN ANALYZE
    -> SELECT TeamName, AvgGoalDifferentialHome, AvgGoalDifferentialAway
    -> FROM (SELECT HomeTeam as TeamName, ((SUM(HomeTeamGoals) - SUM(AwayTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialHome
    ->   FROM MatchGameHistory m
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->  (SELECT AwayTeam as TeamName, ((SUM(AwayTeamGoals) - SUM(HomeTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialAway
    ->   FROM  MatchGameHistory m
    ->   GROUP BY TeamName) sub2;
```

```
| -> Nested loop inner join  (cost=4438.10 rows=0) (actual time=5.761..5.786 rows=17 loops=1)
    -> Filter: (sub1.TeamName is not null)  (cost=0.11..193.08 rows=1694) (actual time=3.595..3.601 rows=17 loops=1)
        -> Table scan on sub1  (cost=2.50..2.50 rows=0) (actual time=3.593..3.597 rows=17 loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.592..3.592 rows=17 loops=1)
                -> Table scan on <temporary>  (actual time=3.554..3.558 rows=17 loops=1)
                    -> Aggregate using temporary table  (actual time=3.552..3.552 rows=17 loops=1)
                        -> Table scan on m  (cost=171.90 rows=1694) (actual time=0.810..1.929 rows=1694 loops=1)
    -> Index lookup on sub2 using <auto_key0> (TeamName=sub1.TeamName)  (actual time=0.128..0.128 rows=1 loops=17)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.157..2.157 rows=17 loops=1)
            -> Table scan on <temporary>  (actual time=2.120..2.123 rows=17 loops=1)
                -> Aggregate using temporary table  (actual time=2.118..2.118 rows=17 loops=1)
                    -> Table scan on m  (cost=171.90 rows=1694) (actual time=0.054..0.609 rows=1694 loops=1)
 |
```

1. Indexing on MatchGameHistory(HomeTeam)

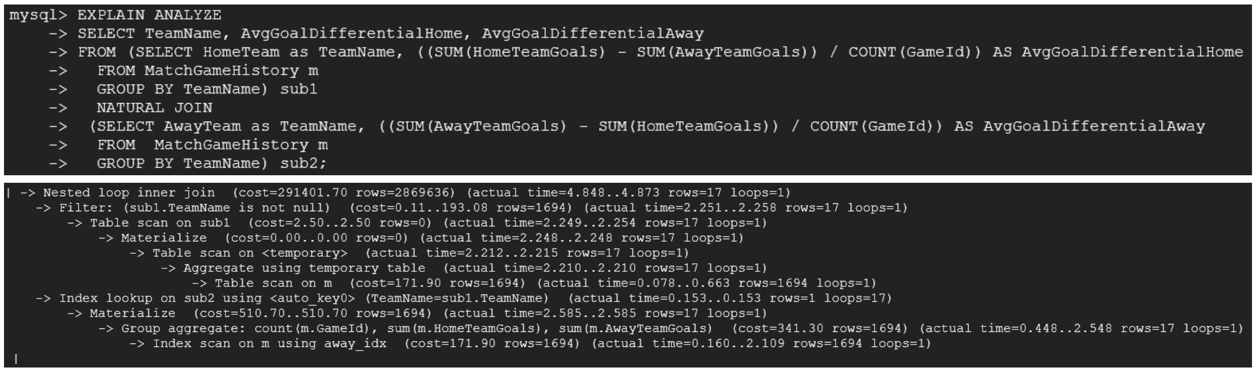


```
mysql> EXPLAIN ANALYZE
    -> SELECT TeamName, AvgGoalDifferentialHome, AvgGoalDifferentialAway
    -> FROM (SELECT HomeTeam as TeamName, ((SUM(HomeTeamGoals) - SUM(AwayTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialHome
    ->   FROM MatchGameHistory m
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->  (SELECT AwayTeam as TeamName, ((SUM(AwayTeamGoals) - SUM(HomeTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialAway
    ->   FROM  MatchGameHistory m
    ->   GROUP BY TeamName) sub2;
```

```
| -> Nested loop inner join  (cost=4438.10 rows=0) (actual time=4.982..5.006 rows=17 loops=1)
    -> Filter: (sub1.TeamName is not null)  (cost=510.51..193.08 rows=1694) (actual time=2.855..2.861 rows=17 loops=1)
        -> Table scan on sub1  (cost=510.71..534.38 rows=1694) (actual time=2.852..2.856 rows=17 loops=1)
            -> Materialize  (cost=510.70..510.70 rows=1694) (actual time=2.848..2.848 rows=17 loops=1)
                -> Group aggregate: count(m.GameId), sum(m.AwayTeamGoals), sum(m.HomeTeamGoals)  (cost=341.30 rows=1694) (actual time=0.647..2.800 rows=17 loops=1)
                    -> Index scan on m using home_idx  (cost=171.90 rows=1694) (actual time=0.240..2.341 rows=1694 loops=1)
    -> Index lookup on sub2 using <auto_key0> (TeamName=sub1.TeamName)  (actual time=0.125..0.126 rows=1 loops=17)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.113..2.113 rows=17 loops=1)
            -> Table scan on <temporary>  (actual time=2.071..2.074 rows=17 loops=1)
                -> Aggregate using temporary table  (actual time=2.070..2.070 rows=17 loops=1)
                    -> Table scan on m  (cost=171.90 rows=1694) (actual time=0.059..0.625 rows=1694 loops=1)
 |
```

   a. No performance improvement detected. This is likely because we are grouping by teams with no exclusion so the group by is performed for all teams. We are also performing aggregate operations on all HomeTeams. Indexing would not help in this case.

2. Indexing on MatchGameHistory(AwayTeam)

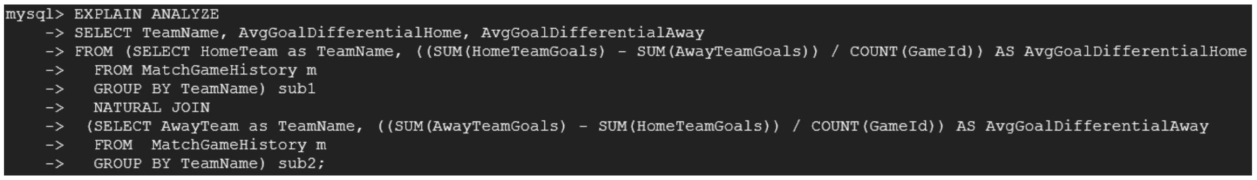


```
mysql> EXPLAIN ANALYZE
    -> SELECT TeamName, AvgGoalDifferentialHome, AvgGoalDifferentialAway
    -> FROM (SELECT HomeTeam as TeamName, ((SUM(HomeTeamGoals) - SUM(AwayTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialHome
    ->   FROM MatchGameHistory m
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->  (SELECT AwayTeam as TeamName, ((SUM(AwayTeamGoals) - SUM(HomeTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialAway
    ->   FROM  MatchGameHistory m
    ->   GROUP BY TeamName) sub2;
```

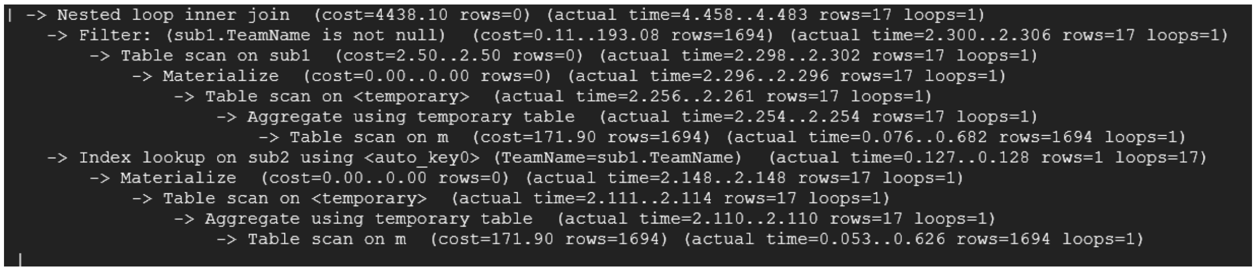
```
| -> Nested loop inner join  (cost=291401.70 rows=2869636) (actual time=4.848..4.873 rows=17 loops=1)
    -> Filter: (sub1.TeamName is not null)  (cost=0.11..193.08 rows=1694) (actual time=2.251..2.258 rows=17 loops=1)
        -> Table scan on sub1  (cost=2.50..2.50 rows=0) (actual time=2.249..2.254 rows=17 loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.248..2.248 rows=17 loops=1)
                -> Table scan on <temporary>  (actual time=2.212..2.215 rows=17 loops=1)
                    -> Aggregate using temporary table  (actual time=2.210..2.210 rows=17 loops=1)
                        -> Table scan on m  (cost=171.90 rows=1694) (actual time=0.078..0.663 rows=1694 loops=1)
    -> Index lookup on sub2 using <auto_key0> (TeamName=sub1.TeamName)  (actual time=0.153..0.153 rows=1 loops=17)
        -> Materialize  (cost=510.70..510.70 rows=1694) (actual time=2.585..2.585 rows=17 loops=1)
            -> Group aggregate: count(m.GameId), sum(m.HomeTeamGoals), sum(m.AwayTeamGoals)  (cost=341.30 rows=1694) (actual time=0.448..2.548 rows=17 loops=1)
                -> Index scan on m using away_idx  (cost=171.90 rows=1694) (actual time=0.160..2.109 rows=1694 loops=1)
 |
```

   a. Significantly worse performance. This is honestly perplexing because we use AwayTeam in almost the exact same fashion as HomeTeam. Perhaps indexing on AwayTeam causes lots of work to be redone somehow as we are joining it with HomeTeam AS TeamName.

3. Indexing on MatchGameHistory(HomeTeamGoals)



```
mysql> EXPLAIN ANALYZE
    -> SELECT TeamName, AvgGoalDifferentialHome, AvgGoalDifferentialAway
    -> FROM (SELECT HomeTeam as TeamName, ((SUM(HomeTeamGoals) - SUM(AwayTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialHome
    ->   FROM MatchGameHistory m
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->  (SELECT AwayTeam as TeamName, ((SUM(AwayTeamGoals) - SUM(HomeTeamGoals)) / COUNT(GameId)) AS AvgGoalDifferentialAway
    ->   FROM  MatchGameHistory m
    ->   GROUP BY TeamName) sub2;
```



```
| -> Nested loop inner join  (cost=4438.10 rows=0) (actual time=4.458..4.483 rows=17 loops=1)
    -> Filter: (sub1.TeamName is not null)  (cost=0.11..193.08 rows=1694) (actual time=2.300..2.306 rows=17 loops=1)
        -> Table scan on sub1  (cost=2.50..2.50 rows=0) (actual time=2.298..2.302 rows=17 loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.296..2.296 rows=17 loops=1)
                -> Table scan on <temporary>  (actual time=2.256..2.261 rows=17 loops=1)
                    -> Aggregate using temporary table  (actual time=2.254..2.254 rows=17 loops=1)
                        -> Table scan on m  (cost=171.90 rows=1694) (actual time=0.076..0.682 rows=1694 loops=1)
    -> Index lookup on sub2 using <auto_key0> (TeamName=sub1.TeamName)  (actual time=0.127..0.128 rows=1 loops=17)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.148..2.148 rows=17 loops=1)
            -> Table scan on <temporary>  (actual time=2.111..2.114 rows=17 loops=1)
                -> Aggregate using temporary table  (actual time=2.110..2.110 rows=17 loops=1)
                    -> Table scan on m  (cost=171.90 rows=1694) (actual time=0.053..0.626 rows=1694 loops=1)
 |
```

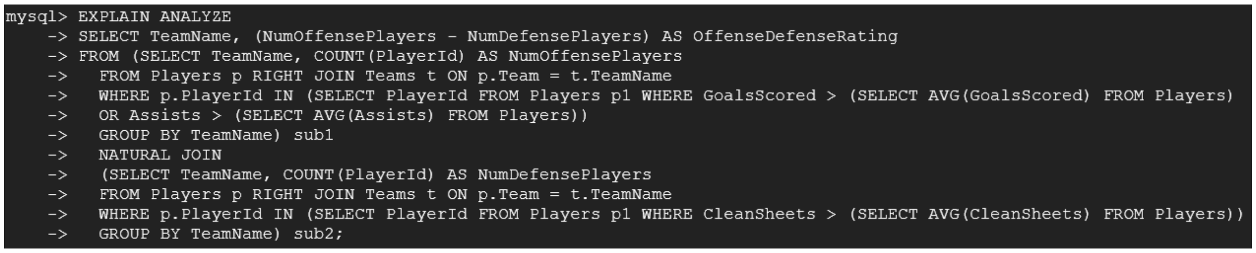
   a. No change. HomeTeamGoals also appears in an aggregate function (SUM) so the entire table is scanned.

4. FINAL SCHEME: No indexing

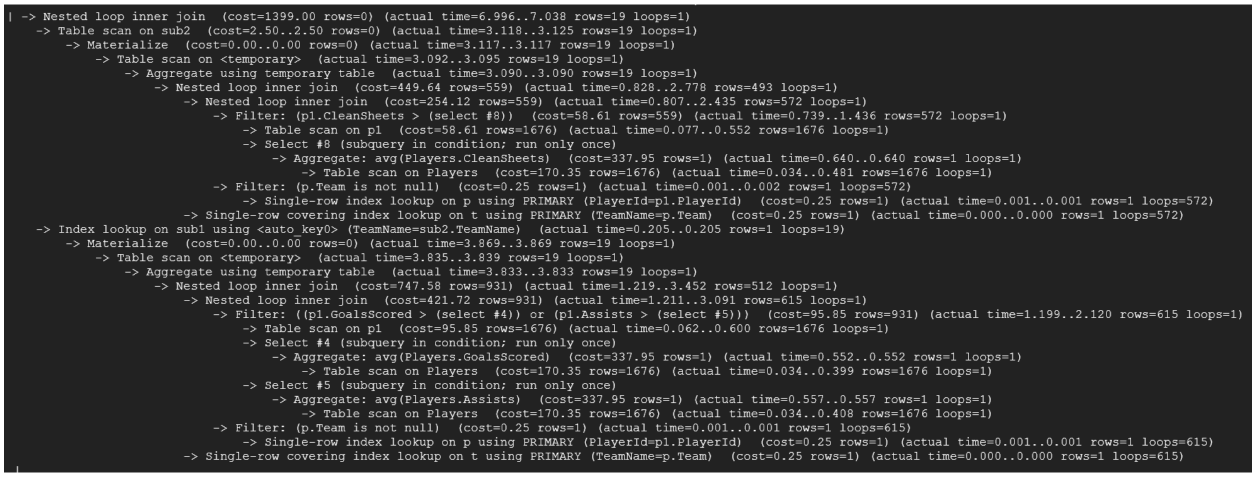
   a. None of our indexing schemes yielded any improvement, and there are no attributes in the query that seem worth trying. The most promising were our VARCHAR attributes, whereas every other attribute is part of an aggregation operation that would scan the entire table.

Query 4

Before Indexing

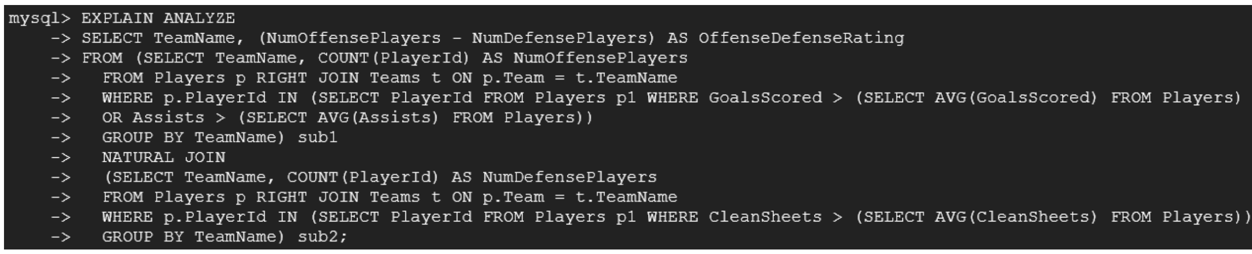


```
mysql> EXPLAIN ANALYZE
    -> SELECT TeamName, (NumOffensePlayers - NumDefensePlayers) AS OffenseDefenseRating
    -> FROM (SELECT TeamName, COUNT(PlayerId) AS NumOffensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   OR Assists > (SELECT AVG(Assists) FROM Players))
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->   (SELECT TeamName, COUNT(PlayerId) AS NumDefensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players))
    ->   GROUP BY TeamName) sub2;
```

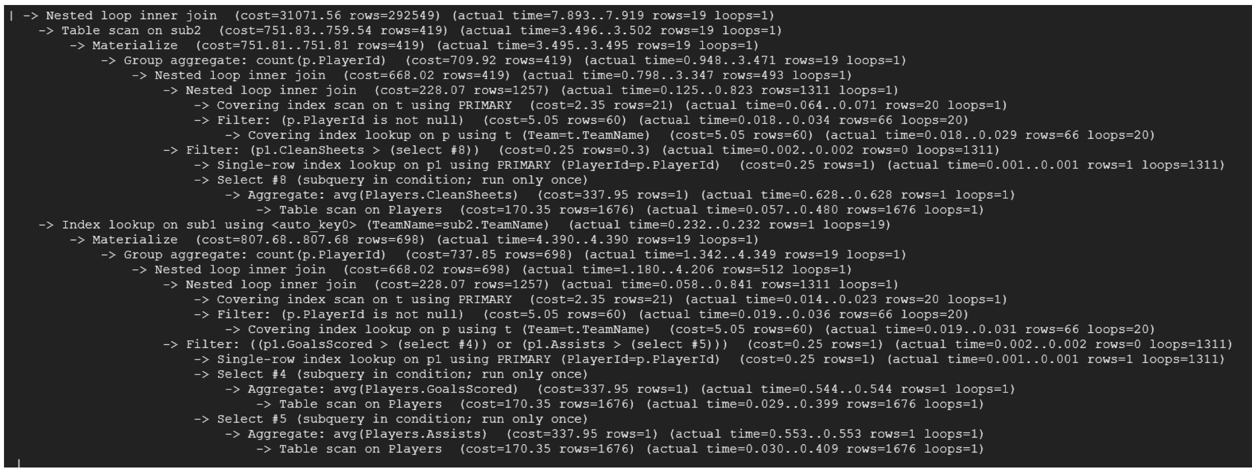


```
| -> Nested loop inner join  (cost=1399.00 rows=0) (actual time=6.996..7.038 rows=19 loops=1)
    -> Table scan on sub2  (cost=2.50..2.50 rows=0) (actual time=3.118..3.125 rows=19 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.117..3.117 rows=19 loops=1)
            -> Table scan on <temporary>  (actual time=3.092..3.095 rows=19 loops=1)
                -> Aggregate using temporary table  (actual time=3.090..3.090 rows=19 loops=1)
                    -> Nested loop inner join  (cost=449.64 rows=559) (actual time=0.828..2.778 rows=493 loops=1)
                        -> Nested loop inner join  (cost=254.12 rows=559) (actual time=0.807..2.435 rows=572 loops=1)
                            -> Filter: (p1.CleanSheets > (select #8))  (cost=58.61 rows=559) (actual time=0.739..1.436 rows=572 loops=1)
                                -> Table scan on p1  (cost=58.61 rows=1676) (actual time=0.077..0.552 rows=1676 loops=1)
                                -> Select #8 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=0.640..0.640 rows=1 loops=1)
                                        -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.034..0.481 rows=1676 loops=1)
                            -> Filter: (p.Team is not null)  (cost=0.25 rows=1) (actual time=0.001..0.002 rows=1 loops=572)
                                -> Single-row index lookup on p using PRIMARY (PlayerId=p1.PlayerId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=572)
                        -> Single-row covering index lookup on t using PRIMARY (TeamName=p.Team)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=572)
    -> Index lookup on sub1 using <auto_key0> (TeamName=sub2.TeamName)  (actual time=0.205..0.205 rows=1 loops=19)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.869..3.869 rows=19 loops=1)
            -> Table scan on <temporary>  (actual time=3.835..3.839 rows=19 loops=1)
                -> Aggregate using temporary table  (actual time=3.833..3.833 rows=19 loops=1)
                    -> Nested loop inner join  (cost=747.58 rows=931) (actual time=1.219..3.452 rows=512 loops=1)
                        -> Nested loop inner join  (cost=421.72 rows=931) (actual time=1.211..3.091 rows=615 loops=1)
                            -> Filter: ((p1.GoalsScored > (select #4)) or (p1.Assists > (select #5)))  (cost=95.85 rows=931) (actual time=1.199..2.120 rows=615 loops=1)
                                -> Table scan on p1  (cost=95.85 rows=1676) (actual time=0.062..0.600 rows=1676 loops=1)
                                -> Select #4 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=0.552..0.552 rows=1 loops=1)
                                        -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.034..0.399 rows=1676 loops=1)
                                -> Select #5 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=0.557..0.557 rows=1 loops=1)
                                        -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.034..0.408 rows=1676 loops=1)
                            -> Filter: (p.Team is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=615)
                                -> Single-row index lookup on p using PRIMARY (PlayerId=p1.PlayerId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=615)
                        -> Single-row covering index lookup on t using PRIMARY (TeamName=p.Team)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=615)
|
```

1. Indexing on Players(Team)



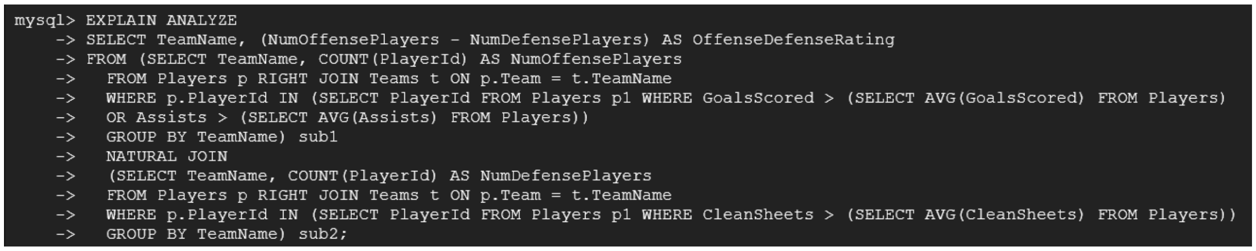
```
mysql> EXPLAIN ANALYZE
    -> SELECT TeamName, (NumOffensePlayers - NumDefensePlayers) AS OffenseDefenseRating
    -> FROM (SELECT TeamName, COUNT(PlayerId) AS NumOffensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   OR Assists > (SELECT AVG(Assists) FROM Players))
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->   (SELECT TeamName, COUNT(PlayerId) AS NumDefensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players))
    ->   GROUP BY TeamName) sub2;
```



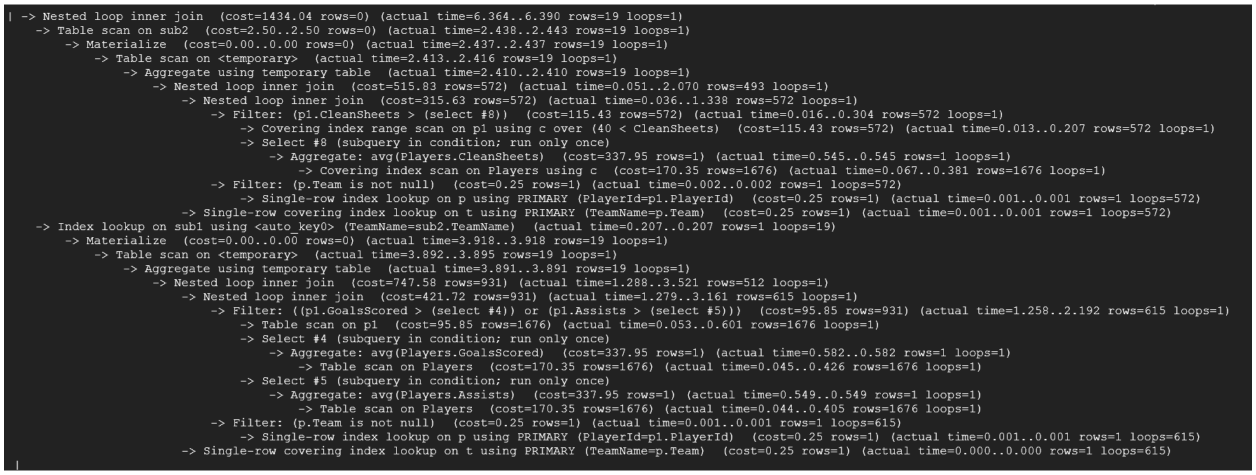
```
| -> Nested loop inner join  (cost=31071.56 rows=292549) (actual time=7.893..7.919 rows=19 loops=1)
    -> Table scan on sub2  (cost=751.83..759.54 rows=419) (actual time=3.496..3.502 rows=19 loops=1)
        -> Materialize  (cost=751.81..751.81 rows=419) (actual time=3.495..3.495 rows=19 loops=1)
            -> Group aggregate: count(p.PlayerId)  (cost=709.92 rows=419) (actual time=0.948..3.471 rows=19 loops=1)
                -> Nested loop inner join  (cost=668.02 rows=419) (actual time=0.798..3.347 rows=493 loops=1)
                    -> Nested loop inner join  (cost=228.07 rows=1257) (actual time=0.125..0.823 rows=1311 loops=1)
                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.064..0.071 rows=20 loops=1)
                        -> Filter: (p.PlayerId is not null)  (cost=5.05 rows=60) (actual time=0.018..0.034 rows=66 loops=20)
                            -> Covering index lookup on p using t (Team=t.TeamName)  (cost=5.05 rows=60) (actual time=0.018..0.029 rows=66 loops=20)
                    -> Filter: (p1.CleanSheets > (select #8))  (cost=0.25 rows=0.3) (actual time=0.002..0.002 rows=0 loops=1311)
                        -> Single-row index lookup on p1 using PRIMARY (PlayerId=p.PlayerId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1311)
                        -> Select #8 (subquery in condition; run only once)
                            -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=0.628..0.628 rows=1 loops=1)
                                -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.057..0.480 rows=1676 loops=1)
    -> Index lookup on sub1 using <auto_key0> (TeamName=sub2.TeamName)  (actual time=0.232..0.232 rows=1 loops=19)
        -> Materialize  (cost=807.68..807.68 rows=698) (actual time=4.390..4.390 rows=19 loops=1)
            -> Group aggregate: count(p.PlayerId)  (cost=737.85 rows=698) (actual time=1.342..4.349 rows=19 loops=1)
                -> Nested loop inner join  (cost=668.02 rows=698) (actual time=1.180..4.206 rows=512 loops=1)
                    -> Nested loop inner join  (cost=228.07 rows=1257) (actual time=0.058..0.841 rows=1311 loops=1)
                        -> Covering index scan on t using PRIMARY  (cost=2.35 rows=21) (actual time=0.014..0.023 rows=20 loops=1)
                        -> Filter: (p.PlayerId is not null)  (cost=5.05 rows=60) (actual time=0.019..0.036 rows=66 loops=20)
                            -> Covering index lookup on p using t (Team=t.TeamName)  (cost=5.05 rows=60) (actual time=0.019..0.031 rows=66 loops=20)
                    -> Filter: ((p1.GoalsScored > (select #4)) or (p1.Assists > (select #5)))  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=0 loops=1311)
                        -> Single-row index lookup on p1 using PRIMARY (PlayerId=p.PlayerId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=1311)
                        -> Select #4 (subquery in condition; run only once)
                            -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=0.544..0.544 rows=1 loops=1)
                                -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.029..0.399 rows=1676 loops=1)
                        -> Select #5 (subquery in condition; run only once)
                            -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=0.553..0.553 rows=1 loops=1)
                                -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.030..0.409 rows=1676 loops=1)
 |
```

   a. Significantly worse performance. Somehow the root-level nested loop inner join accesses hundreds of thousands of rows in this case. The query processor may have been getting stuck on the index.

2. Indexing on Players(CleanSheets)



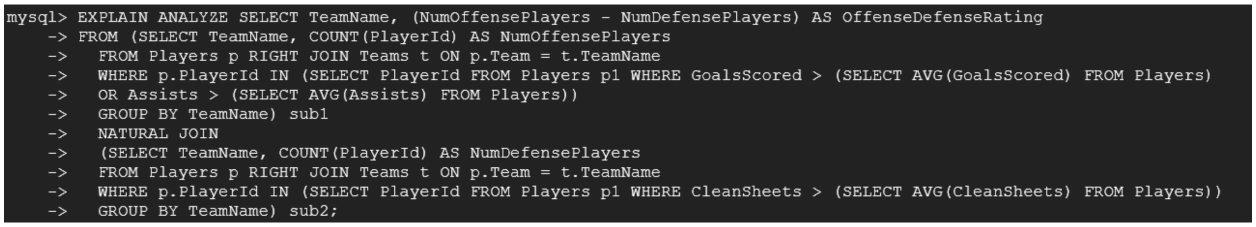
```
mysql> EXPLAIN ANALYZE
    -> SELECT TeamName, (NumOffensePlayers - NumDefensePlayers) AS OffenseDefenseRating
    -> FROM (SELECT TeamName, COUNT(PlayerId) AS NumOffensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   OR Assists > (SELECT AVG(Assists) FROM Players))
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->   (SELECT TeamName, COUNT(PlayerId) AS NumDefensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players))
    ->   GROUP BY TeamName) sub2;
```



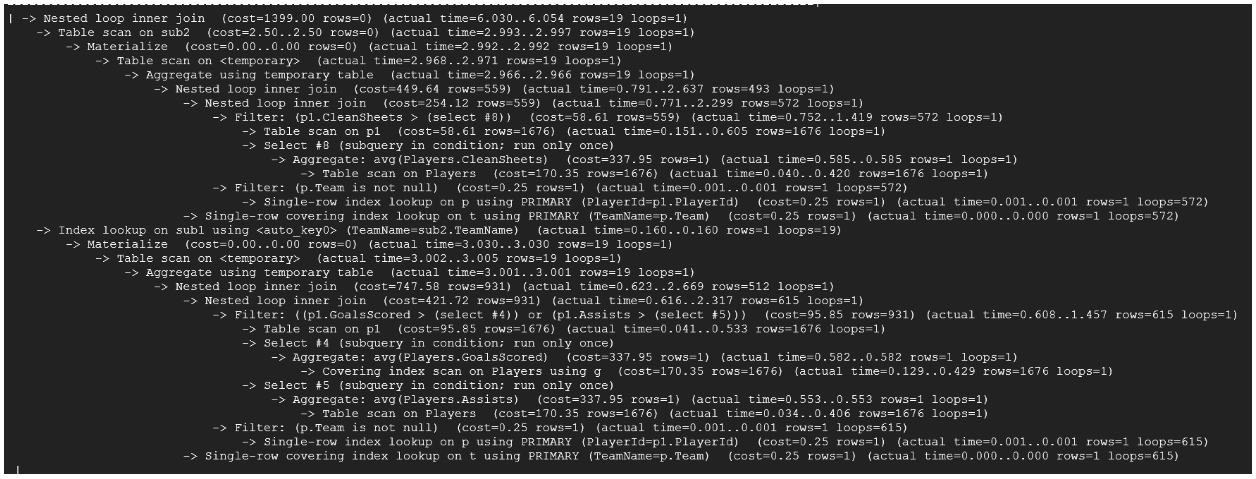
```
| -> Nested loop inner join  (cost=1434.04 rows=0) (actual time=6.364..6.390 rows=19 loops=1)
    -> Table scan on sub2  (cost=2.50..2.50 rows=0) (actual time=2.438..2.443 rows=19 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.437..2.437 rows=19 loops=1)
            -> Table scan on <temporary>  (actual time=2.413..2.416 rows=19 loops=1)
                -> Aggregate using temporary table  (actual time=2.410..2.410 rows=19 loops=1)
                    -> Nested loop inner join  (cost=515.83 rows=572) (actual time=0.051..2.070 rows=493 loops=1)
                        -> Nested loop inner join  (cost=315.63 rows=572) (actual time=0.036..1.338 rows=572 loops=1)
                            -> Filter: (p1.CleanSheets > (select #8))  (cost=115.43 rows=572) (actual time=0.016..0.304 rows=572 loops=1)
                                -> Covering index range scan on p1 using c over (40 < CleanSheets)  (cost=115.43 rows=572) (actual time=0.013..0.207 rows=572 loops=1)
                                -> Select #8 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=0.545..0.545 rows=1 loops=1)
                                        -> Covering index scan on Players using c  (cost=170.35 rows=1676) (actual time=0.067..0.381 rows=1676 loops=1)
                            -> Filter: (p.Team is not null)  (cost=0.25 rows=1) (actual time=0.002..0.002 rows=1 loops=572)
                                -> Single-row index lookup on p using PRIMARY (PlayerId=p1.PlayerId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=572)
                        -> Single-row covering index lookup on t using PRIMARY (TeamName=p.Team)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=572)
    -> Index lookup on sub1 using <auto_key0> (TeamName=sub2.TeamName)  (actual time=0.207..0.207 rows=1 loops=19)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.918..3.918 rows=19 loops=1)
            -> Table scan on <temporary>  (actual time=3.892..3.895 rows=19 loops=1)
                -> Aggregate using temporary table  (actual time=3.891..3.891 rows=19 loops=1)
                    -> Nested loop inner join  (cost=747.58 rows=931) (actual time=1.288..3.521 rows=512 loops=1)
                        -> Nested loop inner join  (cost=421.72 rows=931) (actual time=1.279..3.161 rows=615 loops=1)
                            -> Filter: ((p1.GoalsScored > (select #4)) or (p1.Assists > (select #5)))  (cost=95.85 rows=931) (actual time=1.258..2.192 rows=615 loops=1)
                                -> Table scan on p1  (cost=95.85 rows=1676) (actual time=0.053..0.601 rows=1676 loops=1)
                                -> Select #4 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=0.582..0.582 rows=1 loops=1)
                                        -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.045..0.426 rows=1676 loops=1)
                                -> Select #5 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=0.549..0.549 rows=1 loops=1)
                                        -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.044..0.405 rows=1676 loops=1)
                            -> Filter: (p.Team is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=615)
                                -> Single-row index lookup on p using PRIMARY (PlayerId=p1.PlayerId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=615)
                        -> Single-row covering index lookup on t using PRIMARY (TeamName=p.Team)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=615)
 |
```

   a. Worse performance. On the row where CleanSheets is accessed, it says that an index range scan was used that accessed less rows than a full table scan but cost more. This means that the query processor did not benefit from indexing.

3. Indexing on Players(GoalsScored)



```
mysql> EXPLAIN ANALYZE SELECT TeamName, (NumOffensePlayers - NumDefensePlayers) AS OffenseDefenseRating
    -> FROM (SELECT TeamName, COUNT(PlayerId) AS NumOffensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE GoalsScored > (SELECT AVG(GoalsScored) FROM Players)
    ->   OR Assists > (SELECT AVG(Assists) FROM Players))
    ->   GROUP BY TeamName) sub1
    ->   NATURAL JOIN
    ->   (SELECT TeamName, COUNT(PlayerId) AS NumDefensePlayers
    ->   FROM Players p RIGHT JOIN Teams t ON p.Team = t.TeamName
    ->   WHERE p.PlayerId IN (SELECT PlayerId FROM Players p1 WHERE CleanSheets > (SELECT AVG(CleanSheets) FROM Players))
    ->   GROUP BY TeamName) sub2;
```



```
| -> Nested loop inner join  (cost=1399.00 rows=0) (actual time=6.030..6.054 rows=19 loops=1)
    -> Table scan on sub2  (cost=2.50..2.50 rows=0) (actual time=2.993..2.997 rows=19 loops=1)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=2.992..2.992 rows=19 loops=1)
            -> Table scan on <temporary>  (actual time=2.968..2.971 rows=19 loops=1)
                -> Aggregate using temporary table  (actual time=2.966..2.966 rows=19 loops=1)
                    -> Nested loop inner join  (cost=449.64 rows=559) (actual time=0.791..2.637 rows=493 loops=1)
                        -> Nested loop inner join  (cost=254.12 rows=559) (actual time=0.771..2.299 rows=572 loops=1)
                            -> Filter: (p1.CleanSheets > (select #8))  (cost=58.61 rows=559) (actual time=0.752..1.419 rows=572 loops=1)
                                -> Table scan on p1  (cost=58.61 rows=1676) (actual time=0.151..0.605 rows=1676 loops=1)
                                -> Select #8 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.CleanSheets)  (cost=337.95 rows=1) (actual time=0.585..0.585 rows=1 loops=1)
                                        -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.040..0.420 rows=1676 loops=1)
                            -> Filter: (p.Team is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=572)
                                -> Single-row index lookup on p using PRIMARY (PlayerId=p1.PlayerId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=572)
                        -> Single-row covering index lookup on t using PRIMARY (TeamName=p.Team)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=572)
    -> Index lookup on sub1 using <auto_key0> (TeamName=sub2.TeamName)  (actual time=0.160..0.160 rows=1 loops=19)
        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=3.030..3.030 rows=19 loops=1)
            -> Table scan on <temporary>  (actual time=3.002..3.005 rows=19 loops=1)
                -> Aggregate using temporary table  (actual time=3.001..3.001 rows=19 loops=1)
                    -> Nested loop inner join  (cost=747.58 rows=931) (actual time=0.623..2.669 rows=512 loops=1)
                        -> Nested loop inner join  (cost=421.72 rows=931) (actual time=0.616..2.317 rows=615 loops=1)
                            -> Filter: ((p1.GoalsScored > (select #4)) or (p1.Assists > (select #5)))  (cost=95.85 rows=931) (actual time=0.608..1.457 rows=615 loops=1)
                                -> Table scan on p1  (cost=95.85 rows=1676) (actual time=0.041..0.533 rows=1676 loops=1)
                                -> Select #4 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.GoalsScored)  (cost=337.95 rows=1) (actual time=0.582..0.582 rows=1 loops=1)
                                        -> Covering index scan on Players using g  (cost=170.35 rows=1676) (actual time=0.129..0.429 rows=1676 loops=1)
                                -> Select #5 (subquery in condition; run only once)
                                    -> Aggregate: avg(Players.Assists)  (cost=337.95 rows=1) (actual time=0.553..0.553 rows=1 loops=1)
                                        -> Table scan on Players  (cost=170.35 rows=1676) (actual time=0.034..0.406 rows=1676 loops=1)
                            -> Filter: (p.Team is not null)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=615)
                                -> Single-row index lookup on p using PRIMARY (PlayerId=p1.PlayerId)  (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=615)
                        -> Single-row covering index lookup on t using PRIMARY (TeamName=p.Team)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=615)
 |
```

   a. There is no change to performance despite GoalsScored being used in a similar fashion to CleanSheets. It appears that there is only a single table scan used to compare find both AVG(GoalsScored) and AVG(Assists). However, adding an index on assists did not increase performance either, yielding a total of 1399.00 as well.

4. **FINAL SCHEME**: No indexing

   a. No combination of indices we tried was able to yield performance increases. As such, we chose not to index this query to optimize it.