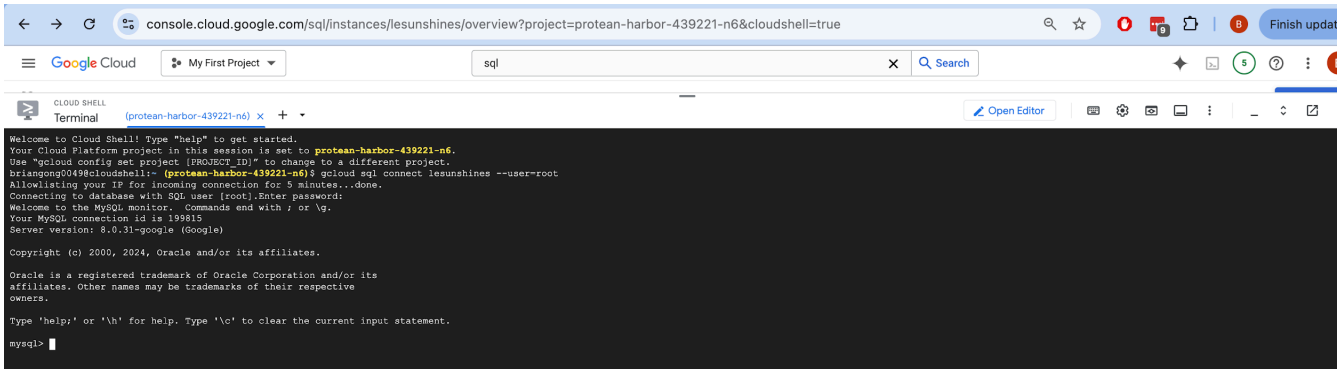
Database Implementation

We chose to implement our tables on GCP. This is our terminal connection login screen.



We have 6 tables: User, Player, Team, Game, Shot, and Season. These are all of their corresponding DDL commands:

```
create table User (
 Username varchar(256),
 Password varchar(256),
 Favorite_Player varchar(256),
 Favorite_Team varchar(256),
 primary key (Username),
 foreign key (Favorite_Player) references Player(PlayerID),
 foreign key (Favorite_Team) references Team(Team_Name)
);
create table Player (
 PlayerID varchar(256),
 Player_Name varchar(256),
 Team_Name varchar(256),
 Season int,
 Position varchar(15),
 Games_Played int,
 Games_Started int,
 Minutes_Played int,
 Field_Goals_Made int,
 Field_Goals_Attempted int,
 Three_Pointers_Attempted int,
 Three_Pointers_Made int,
 Two_Pointers_Attempted int,
 Two_Pointers_Made int,
```

```
  Free_Throws_Made int,
  Free_Throws_Attempted int,
  Offensive_Rebounds int,
  Defensive_Rebounds int,
  Assists int,
  Steals int,
  Blocks int,
  Turnovers int,
  Personal_Fouls int,
  Points int,
  foreign key (Team_Name) references Team(Team_Name),
  foreign key (Season) references Season(Season),
  primary key (PlayerID, Team_Name, Season)
);
  create table Team (
  Team_Name varchar(256),
  Season int,
  Player_IDs varchar(1024),
  primary key (Team_Name, Season)
);
create table Game(
  GameID varchar(256) primary key,
  Date varchar(256),
  Season int,
  Home_Score int,
  Visitor_Score int,
  Home_Team varchar(256),
  Away_Team varchar(256),
  Arena_Name varchar(256),
  foreign key (Season) references Season(Season)
  foreign key (Home_Team) references Team(Team_Name),
  foreign key (Away_Team) references Team(Team_Name)
);
create table Shot (
  GameID varchar(256),
  ShotID int,
  PLAYER_NAME varchar(256),
  Team_Name varchar(256),
  PERIOD int,
```

```
 MINUTES_REMAINING int,
 SECONDS_REMAINING int,
 EVENT_TYPE varchar(256),
 ACTION_TYPE varchar(256),
 SHOT_TYPE varchar(256),
 SHOT_ZONE_BASIC varchar(256),
SHOT_ZONE_AREA varchar(256),
SHOT_ZONE_RANGE varchar(256),
SHOT_DISTANCE int,
LOC_X int,
LOC_Y int,
SHOT_ATTEMPTED_FLAG int,
SHOT_MADE_FLAG int,
GAME_DATE varchar(8),
HTM varchar(256),
VTM varchar(256),
PlayerID varchar(256),
primary key (GameID, ShotID),
foreign key (GameID) references Game(GameID),
foreign key (PlayerID) references Player(PlayerID)
);
create table Season (
 Season int,
 Champion varchar(256),
 Most_Valuable_Player varchar(256),
 Defensive_Player_Of_The_Year varchar(256),
 Rookie_Of_The_Year varchar(256),
 Sixth_Man_Of_The_Year varchar(256),
 Most_Improved_Player varchar(256),
 primary key (Season),
 foreign key (Champion) references Team(Team_Name),
 foreign key (Most_Valuable_Player) references Player(PlayerID),
 foreign key (Defensive_Player_Of_The_Year) references Player(PlayerID),
 foreign key (Rookie_Of_The_Year) references Player(PlayerID),
 foreign key (Sixth_Man_Of_The_Year) references Player(PlayerID),
 foreign key (Most_Improved_Player) references Player(PlayerID)
);
```

These are our row counts for each of our tables. You can see that we have three tables with much more than 1000 rows, with our most being over a million rows.

```
mysql> select count(*) from Team;
+----------+
| count(*) |
+----------+
|      464 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from Game;
+----------+
| count(*) |
+----------+
|    17667 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from Player;
+----------+
| count(*) |
+----------+
|     9455 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from User;
+----------+
| count(*) |
+----------+
|      100 |
+----------+
1 row in set (0.00 sec)

mysql> select count(*) from Season;
+----------+
| count(*) |
+----------+
|       14 |
+----------+
1 row in set (0.03 sec)

mysql>
```

```
mysql> select count(*) from Shot;
+----------+
| count(*) |
+----------+
|  1161223 |
+----------+
1 row in set (0.46 sec)

mysql>
```

Advanced Queries and Indexing Analysis

We have created four advanced queries that provide useful information, and will be incorporated into our final application.

Query 1:

Here is the first query, along with its output (note that we replaced [current user's username] with "user_1", the first test user in our database). The function is to find players that have won season awards and have been on the user's favorite team, to show the user how successful their favorite team has been historically in picking strong players.

```
select Player_Name,
        sum(PlayerID=Most_Valuable_Player)
        +sum(PlayerID=Defensive_Player_Of_The_Year)
        +sum(PlayerID=Rookie_Of_The_Year)
        +sum(PlayerID=Sixth_Man_Of_The_Year)
        +sum(PlayerID=Most_Improved_Player)
        as awards_won
from Player join Season on
        (PlayerID=Most_Valuable_Player)
        or (PlayerID=Defensive_Player_Of_The_Year)
        or (PlayerID=Rookie_Of_The_Year)
        or (PlayerID=Sixth_Man_Of_The_Year)
        or (PlayerID=Most_Improved_Player)
where Player_ID in (
        select distinct PlayerID
        from Player
        where Player.Team_Name = (
                select Favorite_Team
                from User
                where Username = [current user's username]
        )
)
group by PlayerID
order by awards_won desc
limit 15;
```

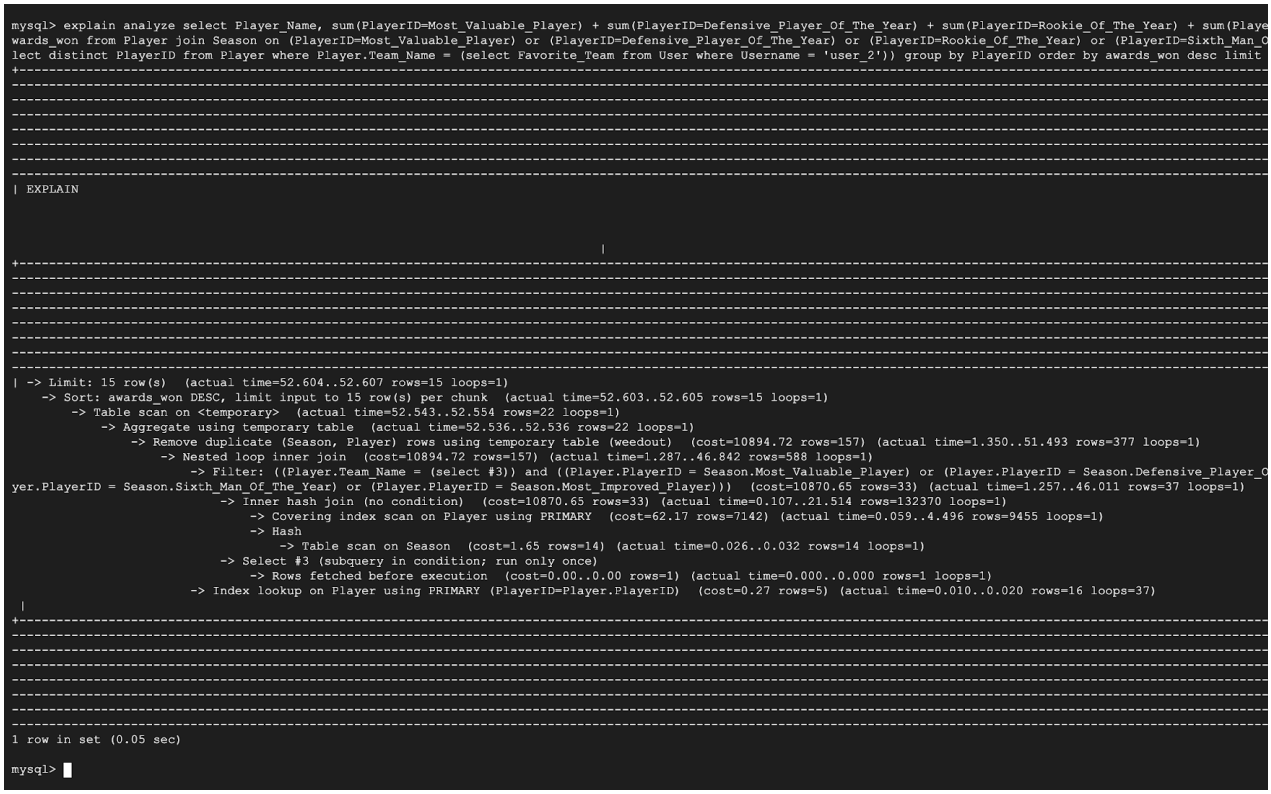
```
mysql> select Player_Name, sum(PlayerID=Most_Valuable_Player) + sum(PlayerID=Defensive_Player_Of
layer join Season on (PlayerID=Most_Valuable_Player) or (PlayerID=Defensive_Player_Of_The_Year)
ayerID from Player where Player.Team_Name = (select Favorite_Team from User where Username = 'us
+------------------------+------------+
| Player_Name            | awards_won |
+------------------------+------------+
| Lou Williams           |         48 |
| James Harden           |         38 |
| Goran Dragić           |         21 |
| Russell Westbrook      |         17 |
| Eric Gordon            |         17 |
| Blake Griffin          |         17 |
| Derrick Rose           |         17 |
| Jimmy Butler           |         16 |
| Kevin Durant           |         16 |
| Kyrie Irving           |         16 |
| Kevin Love             |         16 |
| Jordan Clarkson        |         15 |
| CJ McCollum            |         14 |
| Michael Carter-Williams |         13 |
| Marc Gasol             |         13 |
+------------------------+------------+
15 rows in set (0.06 sec)
```

This is the first query's performance (viewed with EXPLAIN ANALYZE) before adding indexes.

```
mysql> explain analyze select Player_Name, sum(PlayerID=Most_Valuable_Player) + sum(PlayerID=Defensive_Player_Of_The_Year) + sum(PlayerID=Rookie_Of_The_Year) + sum(PlayerID=S
wards_won from Player join Season on (PlayerID=Most_Valuable_Player) or (PlayerID=Defensive_Player_Of_The_Year) or (PlayerID=Rookie_Of_The_Year) or (PlayerID=Sixth_Man_Of_The_
lect distinct PlayerID from Player where Player.Team_Name = (select Favorite_Team from User where Username = 'user_2')) group by PlayerID order by awards_won desc limit 15;
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| EXPLAIN                                                                                                                                                                  |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (actual time=54.721..54.724 rows=15 loops=1)
    -> Sort: awards_won DESC, limit input to 15 row(s) per chunk  (actual time=54.720..54.722 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=54.657..54.667 rows=22 loops=1)
            -> Aggregate using temporary table  (actual time=54.652..54.652 rows=22 loops=1)
                -> Remove duplicate (Season, Player) rows using temporary table (weedout)  (cost=10894.72 rows=157) (actual time=1.371..53.484 rows=377 loops=1)
                    -> Nested loop inner join  (cost=10894.72 rows=157) (actual time=1.308..48.474 rows=588 loops=1)
                        -> Filter: ((Player.Team_Name = (select #3)) and ((Player.PlayerID = Season.Most_Valuable_Player) or (Player.PlayerID = Season.Defensive_Player_Of_The_
yer.PlayerID = Season.Sixth_Man_Of_The_Year) or (Player.PlayerID = Season.Most_Improved_Player)))  (cost=10870.65 rows=33) (actual time=1.273..45.725 rows=37 loops=1)
                            -> Inner hash join (no condition)  (cost=10870.65 rows=33) (actual time=0.099..21.293 rows=132370 loops=1)
                                -> Covering index scan on Player using PRIMARY  (cost=62.17 rows=7142) (actual time=0.041..4.656 rows=9455 loops=1)
                                -> Hash
                                    -> Table scan on Season  (cost=1.65 rows=14) (actual time=0.035..0.040 rows=14 loops=1)
                            -> Select #3 (subquery in condition; run only once)
                                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                        -> Index lookup on Player using PRIMARY (PlayerID=Player.PlayerID)  (cost=0.27 rows=5) (actual time=0.061..0.072 rows=16 loops=37)
 |
+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
1 row in set (0.06 sec)
```

The first thing we tried was an index on the User's Favorite Team for efficient lookup:

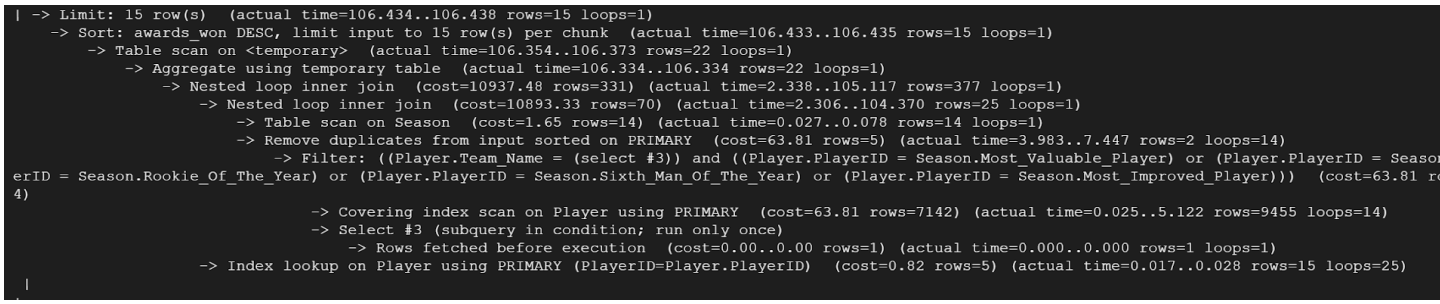
```
CREATE INDEX idx_user_favorite_team ON User(Favorite_Team);
```



There is no marked improvement over the query with no index. The most costly operations from the query are the joins, and this index makes no difference at those due to it being part of a subquery.

The second thing we tried was an index on Player's Team_Name and PlayerID for fast retrieval of players on a specific team:

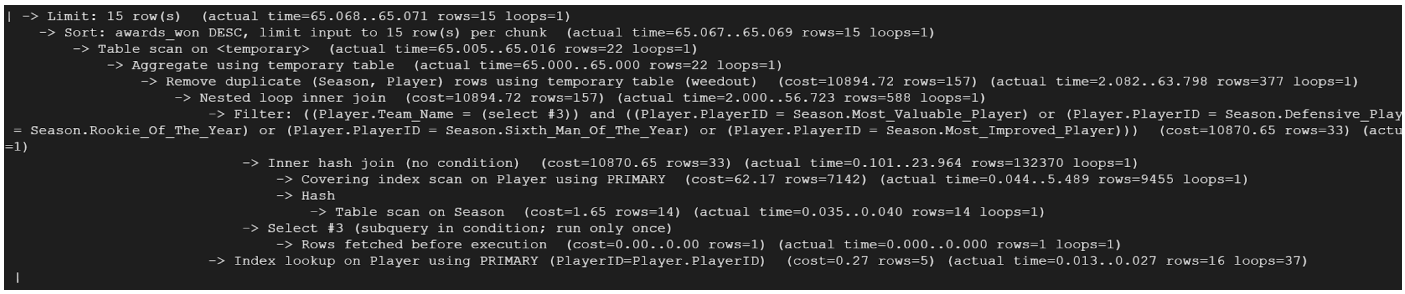
```
CREATE INDEX idx_player_team_name_playerid ON Player(Team_Name, PlayerID);
```



This one actually ran with slightly more cost at each step, although not by much. However, it did not need an inner hash join before index scanning on Player, decreasing the overall cost by about 10,000. This is what we expected, as it can apply the index instead.

The third thing we tried was a composite index on all award columns in the Season table:

```
CREATE INDEX idx_season_awards ON Season(
  Most_Valuable_Player,
  Defensive_Player_Of_The_Year,
  Rookie_Of_The_Year,
  Sixth_Man_Of_The_Year,
  Most_Improved_Player
);
```



```
| -> Limit: 15 row(s)  (actual time=65.068..65.071 rows=15 loops=1)
    -> Sort: awards_won DESC, limit input to 15 row(s) per chunk  (actual time=65.067..65.069 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=65.005..65.016 rows=22 loops=1)
            -> Aggregate using temporary table  (actual time=65.000..65.000 rows=22 loops=1)
                -> Remove duplicate (Season, Player) rows using temporary table (weedout)  (cost=10894.72 rows=157) (actual time=2.082..63.798 rows=377 loops=1)
                    -> Nested loop inner join  (cost=10894.72 rows=157) (actual time=2.000..56.723 rows=588 loops=1)
                        -> Filter: ((Player.Team_Name = (select #3)) and ((Player.PlayerID = Season.Most_Valuable_Player) or (Player.PlayerID = Season.Defensive_Play
 = Season.Rookie_Of_The_Year) or (Player.PlayerID = Season.Sixth_Man_Of_The_Year) or (Player.PlayerID = Season.Most_Improved_Player)))  (cost=10870.65 rows=33) (actu
=1)
                            -> Inner hash join (no condition)  (cost=10870.65 rows=33) (actual time=0.101..23.964 rows=132370 loops=1)
                                -> Covering index scan on Player using PRIMARY  (cost=62.17 rows=7142) (actual time=0.044..5.489 rows=9455 loops=1)
                                -> Hash
                                    -> Table scan on Season  (cost=1.65 rows=14) (actual time=0.035..0.040 rows=14 loops=1)
                            -> Select #3 (subquery in condition; run only once)
                                -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                        -> Index lookup on Player using PRIMARY (PlayerID=Player.PlayerID)  (cost=0.27 rows=5) (actual time=0.013..0.027 rows=16 loops=37)
 |
```

This one had the same costs at each step as the queries without the index and with the index on the favorite team. We expected this to have a similar effect to the index of PlayerID due to it being the other half of the join, but it did not decrease the cost at all.

Ultimately the best performing index was when we indexed Team_Name and PlayerID, due to the subquery checking Team_Name and joining on the PlayerID with the award winner IDs for each season.

Query 2:

Here is the second query, along with its output (note that we replaced [current user's username] with "user_1", the first test user in our database). The function is to get the points scored and shooting% from the last fifteen games of the user's favorite player, to show the recent trend in how their favorite player is performing.

```
select sum(Shot.SHOT_MADE_FLAG) * 2 as points_scored,
avg(Shot.SHOT_MADE_FLAG) as shooting_percentage, GameID
from Shot join Game using(GameID)
where Shot.GameID in (
        select GameID
        from Game
        where Home_Team in (
                select Team_Name
                from Player
                where PlayerID = (
                        select Favorite_Player
                        from User
                        where Username = [current user's username]
                )
        )
        or Away_Team in (
                select Team_Name
                from Player
                where PlayerID = (
                        select Favorite_Player
                        from User
                        where Username = [current user's username]
                )
        )
) and PlayerID = (
        select Favorite_Player
        from User
        where Username = [current user's username]
)
group by Shot.GameID
order by Shot.GAME_DATE desc
limit 15;
```

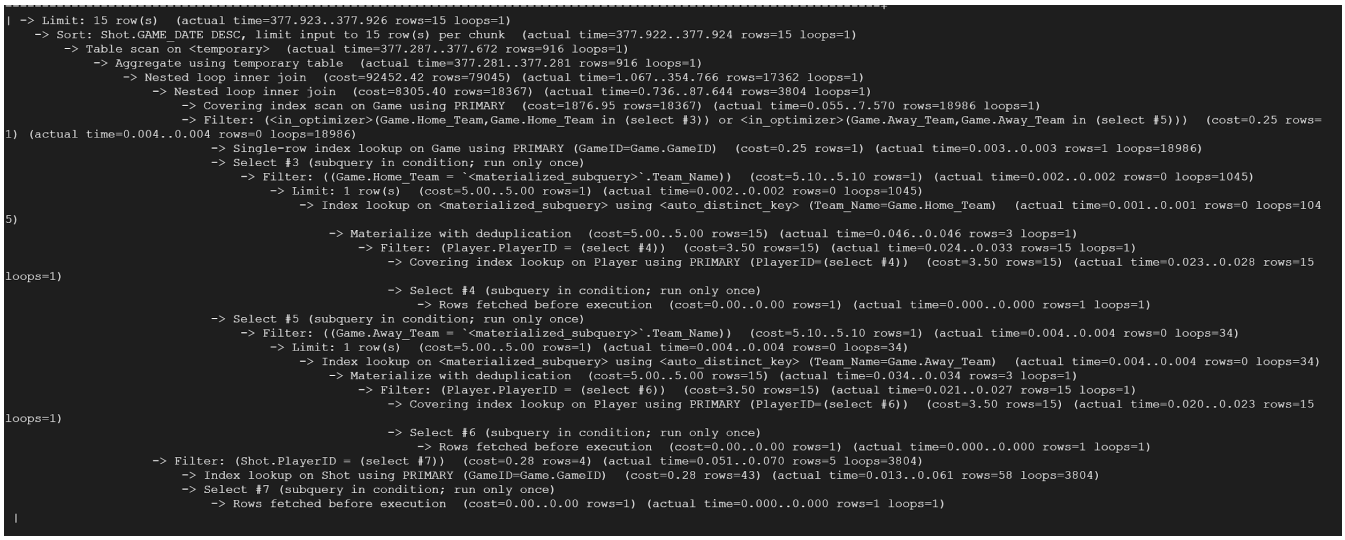
```
mysql> select sum(Shot.SHOT_MADE_FLAG) * 2 as points_scored, avg(Shot.SHOT_MADE_FLAG) as shooting_pe
 GameID from Game where Home_Team in (select Team_Name from Player where PlayerID = (select Favorite
 from Player where PlayerID = (select Favorite_Player from User where Username = 'user_1'))) and Pla
hot.GameID order by Shot.GAME_DATE desc limit 15;
+---------------+---------------------+--------------------+
| points_scored | shooting_percentage | GameID             |
+---------------+---------------------+--------------------+
|            22 |              0.5500 | LAL-NOP-2024-04-14 |
|            26 |              0.6500 | LAL-MEM-2024-04-12 |
|            28 |              0.6364 | GSW-LAL-2024-04-09 |
|            20 |              0.5556 | CLE-LAL-2024-04-06 |
|            18 |              0.5000 | LAL-WAS-2024-04-03 |
|            20 |              0.8333 | LAL-TOR-2024-04-02 |
|            26 |              0.7647 | LAL-BKN-2024-03-31 |
|            12 |              0.5000 | LAL-IND-2024-03-29 |
|            16 |              0.5714 | LAL-MEM-2024-03-27 |
|            16 |              0.4211 | IND-LAL-2024-03-24 |
|            14 |              0.4667 | PHI-LAL-2024-03-22 |
|            20 |              0.7143 | ATL-LAL-2024-03-18 |
|            30 |              0.6522 | GSW-LAL-2024-03-16 |
|            12 |              0.3750 | LAL-SAC-2024-03-13 |
|            20 |              0.6250 | MIN-LAL-2024-03-10 |
+---------------+---------------------+--------------------+
```

This is the second query's performance (viewed with EXPLAIN ANALYZE) before adding indexes.

```
| -> Limit: 15 row(s)  (actual time=357.433..357.435 rows=15 loops=1)
    -> Sort: Shot.GAME_DATE DESC, limit input to 15 row(s) per chunk  (actual time=357.432..357.433 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=356.939..357.261 rows=916 loops=1)
            -> Aggregate using temporary table  (actual time=356.935..356.935 rows=916 loops=1)
                -> Nested loop inner join  (cost=92452.42 rows=79045) (actual time=1.014..335.296 rows=17362 loops=1)
                    -> Nested loop inner join  (cost=8305.40 rows=18367) (actual time=0.687..79.004 rows=3804 loops=1)
                        -> Covering index scan on Game using PRIMARY  (cost=1876.95 rows=18367) (actual time=0.053..6.442 rows=18986 loops=1)
                        -> Filter: (<in_optimizer>(Game.Home_Team,Game.Home_Team in (select #3)) or <in_optimizer>(Game.Away_Team,Game.Away_Team in (select #5)))  (cost=0.25 rows=
1) (actual time=0.004..0.004 rows=0 loops=18986)
                            -> Single-row index lookup on Game using PRIMARY (GameID=Game.GameID)  (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=18986)
                            -> Select #3 (subquery in condition; run only once)
                                -> Filter: ((Game.Home_Team = `<materialized_subquery>`.Team_Name))  (cost=5.10..5.10 rows=1) (actual time=0.002..0.002 rows=0 loops=1045)
                                    -> Limit: 1 row(s)  (cost=5.00..5.00 rows=1) (actual time=0.001..0.001 rows=0 loops=1045)
                                        -> Index lookup on <materialized_subquery> using <auto_distinct_key> (Team_Name=Game.Home_Team)  (actual time=0.001..0.001 rows=0 loops=104
5)
                                            -> Materialize with deduplication  (cost=5.00..5.00 rows=15) (actual time=0.064..0.064 rows=3 loops=1)
                                                -> Filter: (Player.PlayerID = (select #4))  (cost=3.50 rows=15) (actual time=0.040..0.049 rows=15 loops=1)
                                                    -> Covering index lookup on Player using PRIMARY (PlayerID=(select #4))  (cost=3.50 rows=15) (actual time=0.039..0.044 rows=15
loops=1)
                                                    -> Select #4 (subquery in condition; run only once)
                                                        -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                            -> Select #5 (subquery in condition; run only once)
                                -> Filter: ((Game.Away_Team = `<materialized_subquery>`.Team_Name))  (cost=5.10..5.10 rows=1) (actual time=0.005..0.005 rows=0 loops=34)
                                    -> Limit: 1 row(s)  (cost=5.00..5.00 rows=1) (actual time=0.004..0.004 rows=0 loops=34)
                                        -> Index lookup on <materialized_subquery> using <auto_distinct_key> (Team_Name=Game.Away_Team)  (actual time=0.004..0.004 rows=0 loops=34)
                                            -> Materialize with deduplication  (cost=5.00..5.00 rows=15) (actual time=0.033..0.033 rows=3 loops=1)
                                                -> Filter: (Player.PlayerID = (select #6))  (cost=3.50 rows=15) (actual time=0.020..0.026 rows=15 loops=1)
                                                    -> Covering index lookup on Player using PRIMARY (PlayerID=(select #6))  (cost=3.50 rows=15) (actual time=0.019..0.021 rows=15
loops=1)
                                                    -> Select #6 (subquery in condition; run only once)
                                                        -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
                    -> Filter: (Shot.PlayerID = (select #7))  (cost=0.28 rows=4) (actual time=0.049..0.067 rows=5 loops=3804)
                        -> Index lookup on Shot using PRIMARY (GameID=Game.GameID)  (cost=0.28 rows=43) (actual time=0.013..0.058 rows=58 loops=3804)
                        -> Select #7 (subquery in condition; run only once)
                            -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
 |
```

The first thing we tried was an index on User's Favorite Player for efficient lookup:

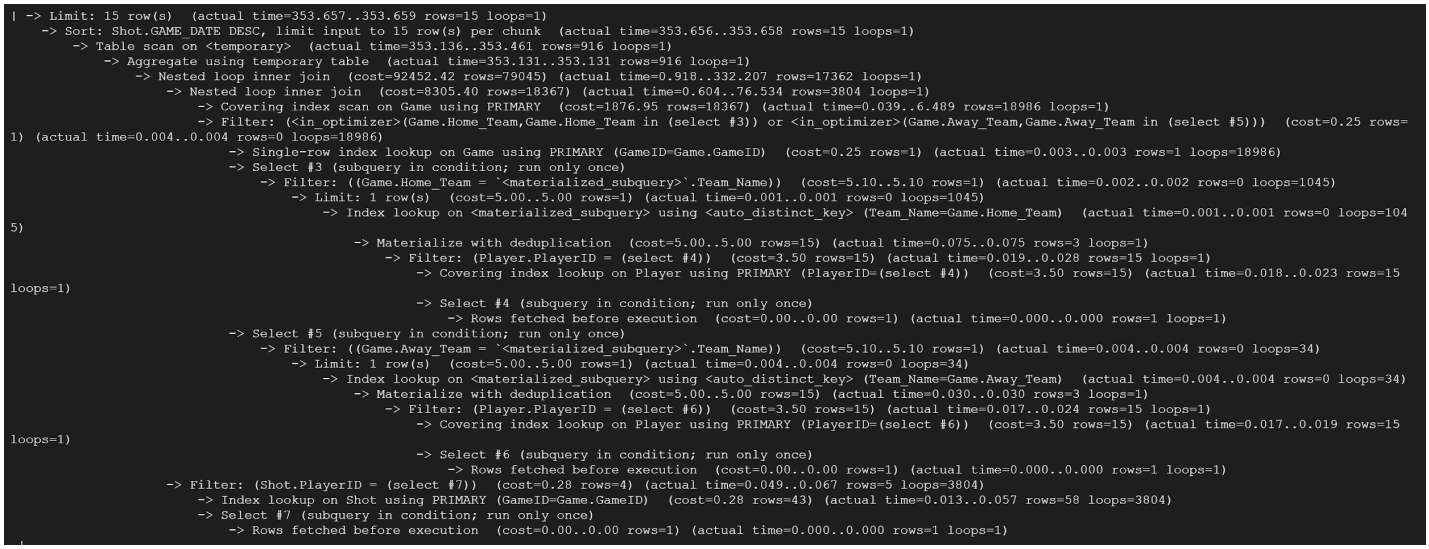
```
CREATE INDEX idx_user_favorite_player ON User(Favorite_Player);
```



There were no major or even minor changes for any of the costs of the operations with this index.

The second thing we tried was a composite index on Game for efficient filtering by team and date:

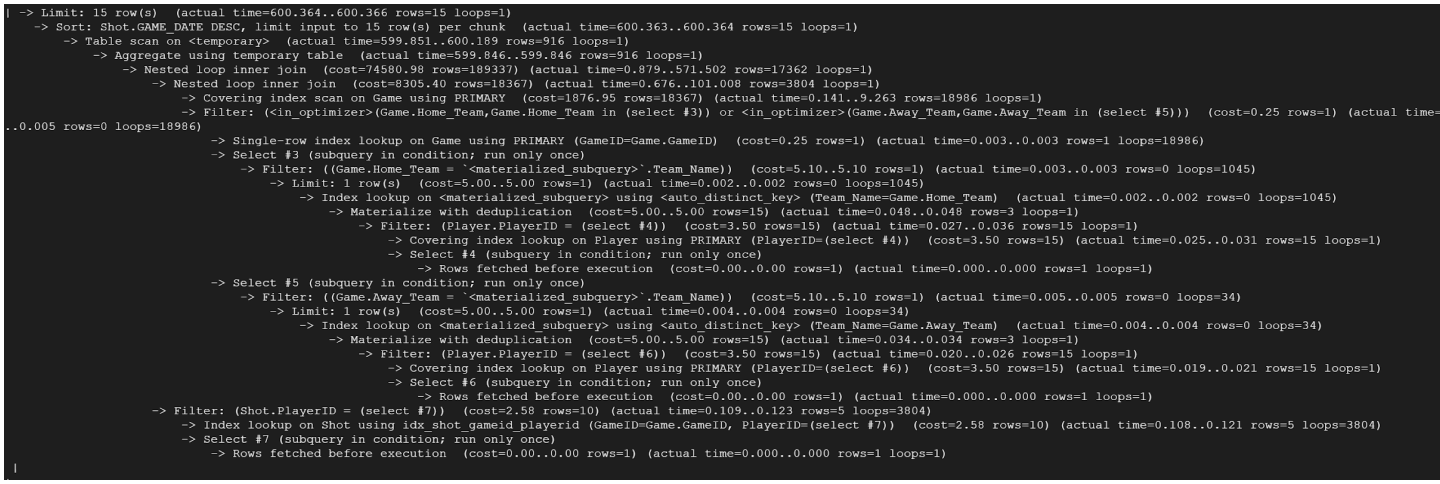
CREATE INDEX idx_game_team_date ON Game(GameID, Home_Team, Away_Team, Date);



There were no major or even minor changes for any of the costs of the operations with this index.

The third thing we tried was a composite index on Shot table for efficient filtering by GameID and PlayerID:

CREATE INDEX idx_shot_gameid_playerid ON Shot(GameID, PlayerID, SHOT_MADE_FLAG);



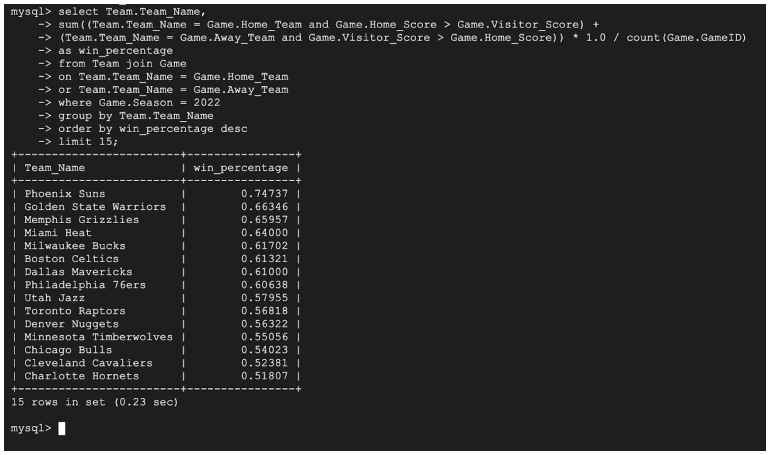
Here we see a cost decrease from 92452 to 74580.98 on the first inner join, but nothing much elsewhere. This is what we expected to see initially with the others given that GameID is what we use to join the two tables, but given the repeated use of PlayerID on the subqueries, along with the sum and avg of SHOT_MADE_FLAG being based on the group by, we expected improvements from those. Percentage wise, it's not the biggest decrease but it is enough to make a difference

Ultimately we went with our third indexing design, which was the composite index on Shot table for filtering by GameID and PlayerID, as it was the only one that provided any cost gains.

Query 3:

Here is the third query, along with its output (note that we replaced [specified season] with 2022). The function is to find the teams with the best records (win percentage) for a given season, which can be compared with their current season performance or extended across seasons.

```
select Team.Team_Name,
        sum((Team.Team_Name = Game.Home_Team and Game.Home_Score >
Game.Visitor_Score) +
        (Team.Team_Name = Game.Away_Team and Game.Visitor_Score >
Game.Home_Score)) * 1.0 / count(Game.GameID)
        as win_percentage
from Team join Game
        on Team.Team_Name = Game.Home_Team
        or Team.Team_Name = Game.Away_Team
where Game.Season = [specified season]
group by Team.Team_Name
order by win_percentage desc
limit 15;
```



```
mysql> select Team.Team_Name,
    -> sum((Team.Team_Name = Game.Home_Team and Game.Home_Score > Game.Visitor_Score) +
    -> (Team.Team_Name = Game.Away_Team and Game.Visitor_Score > Game.Home_Score)) * 1.0 / count(Game.GameID)
    -> as win_percentage
    -> from Team join Game
    -> on Team.Team_Name = Game.Home_Team
    -> or Team.Team_Name = Game.Away_Team
    -> where Game.Season = 2022
    -> group by Team.Team_Name
    -> order by win_percentage desc
    -> limit 15;
+-----------------------+----------------+
| Team_Name             | win_percentage |
+-----------------------+----------------+
| Phoenix Suns          |        0.74737 |
| Golden State Warriors |        0.66346 |
| Memphis Grizzlies     |        0.65957 |
| Miami Heat            |        0.64000 |
| Milwaukee Bucks       |        0.61702 |
| Boston Celtics        |        0.61321 |
| Dallas Mavericks      |        0.61000 |
| Philadelphia 76ers    |        0.60638 |
| Utah Jazz             |        0.57955 |
| Toronto Raptors       |        0.56818 |
| Denver Nuggets        |        0.56322 |
| Minnesota Timberwolves |       0.55056 |
| Chicago Bulls         |        0.54023 |
| Cleveland Cavaliers   |        0.52381 |
| Charlotte Hornets     |        0.51807 |
+-----------------------+----------------+
15 rows in set (0.23 sec)

mysql>
```

This is the third query's performance (viewed with EXPLAIN ANALYZE) before adding indexes.

```
mysql> explain analyze select Team.Team_Name,
    -> sum((Team.Team_Name = Game.Home_Team and Game.Home_Score > Game.Visitor_Score) +
    -> (Team.Team_Name = Game.Away_Team and Game.Visitor_Score > Game.Home_Score)) * 1.0 / count(Game.GameID)
    -> as win_percentage
    -> from Team join Game
    -> on Team.Team_Name = Game.Home_Team
    -> or Team.Team_Name = Game.Away_Team
    -> where Game.Season = 2022
    -> group by Team.Team_Name
    -> order by win_percentage desc
    -> limit 15;
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
| EXPLAIN



                                                                             |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (actual time=231.458..231.460 rows=15 loops=1)
    -> Sort: win_percentage DESC, limit input to 15 row(s) per chunk  (actual time=231.456..231.457 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=231.394..231.409 rows=30 loops=1)
            -> Aggregate using temporary table  (actual time=231.390..231.390 rows=30 loops=1)
                -> Filter: ((Team.Team_Name = Game.Home_Team) or (Team.Team_Name = Game.Away_Team))  (cost=80292.72 rows=3379) (actual time=10.1
                    -> Inner hash join (no condition)  (cost=80292.72 rows=3379) (actual time=10.100..70.657 rows=613872 loops=1)
                        -> Covering index scan on Team using PRIMARY  (cost=0.03 rows=464) (actual time=0.131..0.669 rows=464 loops=1)
                        -> Hash
                            -> Filter: (Game.Season = 2022)  (cost=1763.45 rows=1691) (actual time=0.137..9.724 rows=1323 loops=1)
                                -> Table scan on Game  (cost=1763.45 rows=16912) (actual time=0.129..8.613 rows=18986 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
1 row in set (0.23 sec)

mysql>
```

The first thing we tried was an index on User's Favorite Player for efficient lookup:

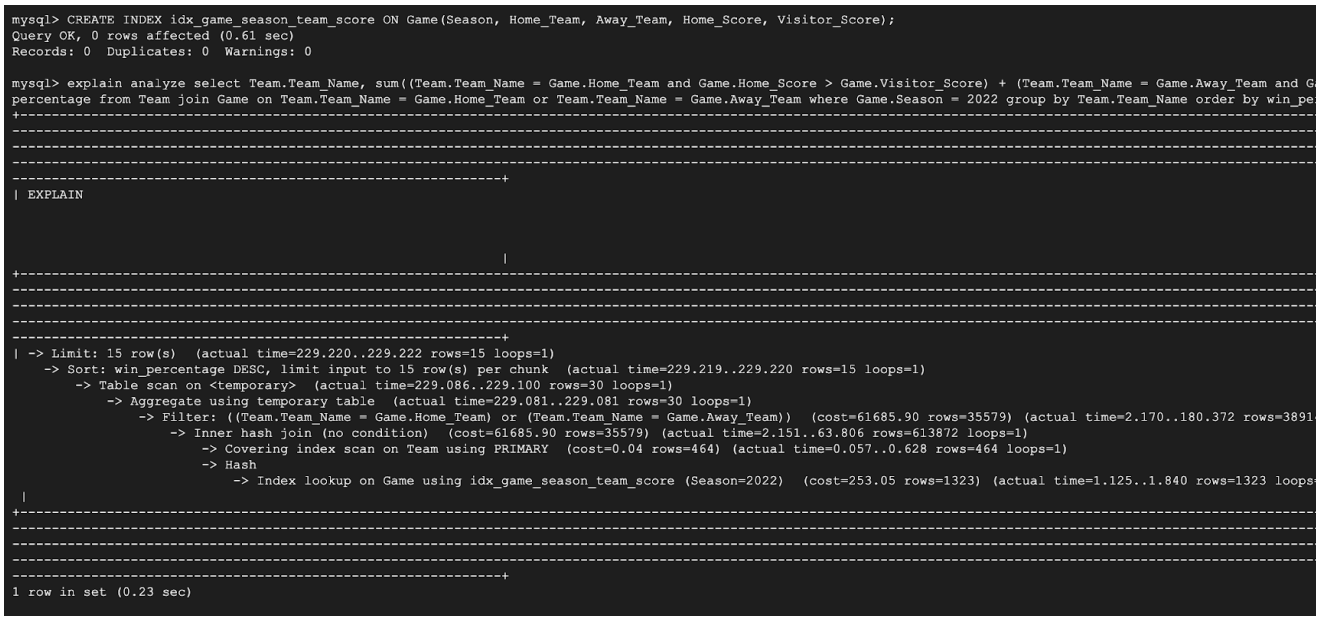
CREATE INDEX idx_user_favorite_player ON User(Favorite_Player);

```
mysql> explain analyze select Team.Team_Name, sum((Team.Team_Name = Game.Home_Team and Game.Home_Score > Game.Visitor_Score) + (Team.Team_Name = Game.
percentage from Team join Game on Team.Team_Name = Game.Home_Team or Team.Team_Name = Game.Away_Team where Game.Season = 2022 group by Team.Team_Name
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
| EXPLAIN



                                                                                   |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
| -> Limit: 15 row(s)  (actual time=224.930..224.932 rows=15 loops=1)
    -> Sort: win_percentage DESC, limit input to 15 row(s) per chunk  (actual time=224.929..224.930 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=224.876..224.889 rows=30 loops=1)
            -> Aggregate using temporary table  (actual time=224.872..224.872 rows=30 loops=1)
                -> Filter: ((Team.Team_Name = Game.Home_Team) or (Team.Team_Name = Game.Away_Team))  (cost=87162.14 rows=49394) (actual time=9.391..17
                    -> Inner hash join (no condition)  (cost=87162.14 rows=49394) (actual time=9.374..68.465 rows=613872 loops=1)
                        -> Covering index scan on Team using PRIMARY  (cost=0.04 rows=464) (actual time=0.053..0.422 rows=464 loops=1)
                        -> Hash
                            -> Filter: (Game.Season = 2022)  (cost=1876.95 rows=1837) (actual time=0.078..9.120 rows=1323 loops=1)
                                -> Table scan on Game  (cost=1876.95 rows=18367) (actual time=0.072..7.948 rows=18986 loops=1)
 |
+------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
1 row in set (0.22 sec)
```

This index actually created an increase in cost from 80292 to 87162 for the first filter and inner hash join, and an increase from 1763 to 1876 for the second filter of Game.Season and the table scan of game

The second thing we tried was a composite index on Game table for efficient season-based filtering and score comparison:

CREATE INDEX idx_game_season_team_score ON Game(Season, Home_Team, Away_Team, Home_Score, Visitor_Score);

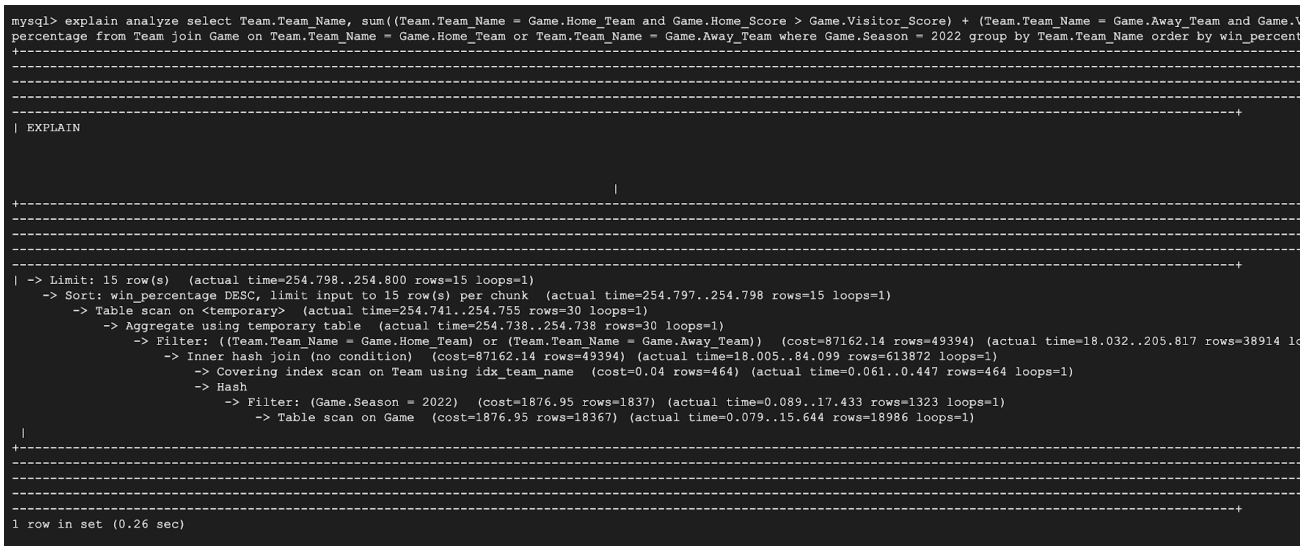


```
mysql> CREATE INDEX idx_game_season_team_score ON Game(Season, Home_Team, Away_Team, Home_Score, Visitor_Score);
Query OK, 0 rows affected (0.61 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain analyze select Team.Team_Name, sum((Team.Team_Name = Game.Home_Team and Game.Home_Score > Game.Visitor_Score) + (Team.Team_Name = Game.Away_Team and G
percentage from Team join Game on Team.Team_Name = Game.Home_Team or Team.Team_Name = Game.Away_Team where Game.Season = 2022 group by Team.Team_Name order by win_pe
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------+
| EXPLAIN

                                                                                                                                                                |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=229.220..229.222 rows=15 loops=1)
    -> Sort: win_percentage DESC, limit input to 15 row(s) per chunk  (actual time=229.219..229.220 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=229.086..229.100 rows=30 loops=1)
            -> Aggregate using temporary table  (actual time=229.081..229.081 rows=30 loops=1)
                -> Filter: ((Team.Team_Name = Game.Home_Team) or (Team.Team_Name = Game.Away_Team))  (cost=61685.90 rows=35579) (actual time=2.170..180.372 rows=3891
                    -> Inner hash join (no condition)  (cost=61685.90 rows=35579) (actual time=2.151..63.806 rows=613872 loops=1)
                        -> Covering index scan on Team using PRIMARY  (cost=0.04 rows=464) (actual time=0.057..0.628 rows=464 loops=1)
                        -> Hash
                            -> Index lookup on Game using idx_game_season_team_score (Season=2022)  (cost=253.05 rows=1323) (actual time=1.125..1.840 rows=1323 loops
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------+
1 row in set (0.23 sec)
```

This one had dramatically improved costs, with the first filter and inner hash join dropping from 80292 to 61685. The Game_table index removed the need for a scan and the filter, dropping the cost of that from 3526 combined to 253.05.

The third thing we tried was a basic index on Team's Team_Name for efficient grouping and lookup:

CREATE INDEX idx_team_name ON Team(Team_Name);



```
mysql> explain analyze select Team.Team_Name, sum((Team.Team_Name = Game.Home_Team and Game.Home_Score > Game.Visitor_Score) + (Team.Team_Name = Game.Away_Team and Game.V
percentage from Team join Game on Team.Team_Name = Game.Home_Team or Team.Team_Name = Game.Away_Team where Game.Season = 2022 group by Team.Team_Name order by win_percent
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------+
| EXPLAIN

                                                                                                                                                                |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=254.798..254.800 rows=15 loops=1)
    -> Sort: win_percentage DESC, limit input to 15 row(s) per chunk  (actual time=254.797..254.798 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=254.741..254.755 rows=30 loops=1)
            -> Aggregate using temporary table  (actual time=254.738..254.738 rows=30 loops=1)
                -> Filter: ((Team.Team_Name = Game.Home_Team) or (Team.Team_Name = Game.Away_Team))  (cost=87162.14 rows=49394) (actual time=18.032..205.817 rows=38914 lo
                    -> Inner hash join (no condition)  (cost=87162.14 rows=49394) (actual time=18.005..84.099 rows=613872 loops=1)
                        -> Covering index scan on Team using idx_team_name  (cost=0.04 rows=464) (actual time=0.061..0.447 rows=464 loops=1)
                        -> Hash
                            -> Filter: (Game.Season = 2022)  (cost=1876.95 rows=1837) (actual time=0.089..17.433 rows=1323 loops=1)
                                -> Table scan on Game  (cost=1876.95 rows=18367) (actual time=0.079..15.644 rows=18986 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------+
1 row in set (0.26 sec)
```

This index was like the first one, meaning that they both performed worse than the base indexing that mysql has.

The game table index with season, teams, and scores was the best performing index by quite a bit, especially considering that the other two actually made the cost higher than the default indexing, so we went with index # 2.

Query 4:

Here is the fourth query, along with its output (note that we replaced [specified season] with 2015 and [current user's username] with "user_1", the first test user in our database). The function is to find the average efficiency of bench players (started in half or less of games played) for each position on the user's favorite team, in a given season. Average efficiency is a stat that is essentially calculated by subtracting mistakes (missed shots, turnovers, etc) from good moves (made shots, rebounds, etc) and averaging it across games. This provides insight into how reliable the bench players are. There are only five output rows because there are only five positions in basketball.

```
select Player.Position,
        avg((Points + Offensive_Rebounds + Defensive_Rebounds + Assists + Steals
+ Blocks
        - (Field_Goals_Attempted - Field_Goals_Made)
        - (Free_Throws_Attempted - Free_Throws_Made) - Turnovers) * 1.0
        / Games_Played)
        as avg_efficiency
from Player
where Player.Season = [specified season]
        and Player.Games_Started <= 0.5 * Player.Games_Played
        and Player.Team_Name = (
                select Favorite_Team
                from User
                where Username = [current user's username]
        )
group by Player.Position
order by avg_efficiency desc;
```

```
mysql> select Player.Position, avg((Points + Offensive_Rebo
e_Throws_Made) - Turnovers) * 1.0 / Games_Played) as avg_ef
ect Favorite_Team from User where User.Username = 'user_1')
+----------+----------------+
| Position | avg_efficiency |
+----------+----------------+
| SF       |   10.053330000 |
| PF       |    7.233330000 |
| PG       |    5.394443333 |
| C        |    5.234850000 |
| SG       |    3.667480000 |
+----------+----------------+
5 rows in set (0.01 sec)

mysql>
```

This is the fourth query's performance (viewed with EXPLAIN ANALYZE) before adding indexes.

```
mysql> EXPLAIN ANALYZE select Player.Position, avg((Points + Offensive_Rebounds + Defensive_Rebounds + Assists + Steals + Blocks - (Field_Goals_Attempted - Field_Goals_Made) - (Free_Throws
_Attempted - Free_Throws_Made) - Turnovers) * 1.0 / Games_Played) as avg_efficiency from Player where Player.Season = 2015 and Player.Games_Started <= 0.5 * Player.Games_Played and Player.
Team_Name = (select Favorite_Team from User where User.Username = 'user_1') group by Player.Position order by avg_efficiency desc;
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
| EXPLAIN


                                                                                                                                                                                  |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
| -> Sort: avg_efficiency DESC  (actual time=10.166..10.166 rows=5 loops=1)
    -> Table scan on <temporary>  (actual time=10.138..10.139 rows=5 loops=1)
        -> Aggregate using temporary table  (actual time=10.135..10.135 rows=5 loops=1)
            -> Filter: ((Player.Team_Name = (select #2)) and (Player.Season = 2015) and (Player.Games_Started <= (0.5 * Player.Games_Played)))  (cost=1000.85 rows=32) (actual time=3.653..1
0.040 rows=12 loops=1)
                -> Table scan on Player  (cost=1000.85 rows=9606) (actual time=0.068..8.446 rows=9455 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
--+
1 row in set (0.01 sec)
```

The first thing we tried was an index on User's Favorite Team for efficient lookup:

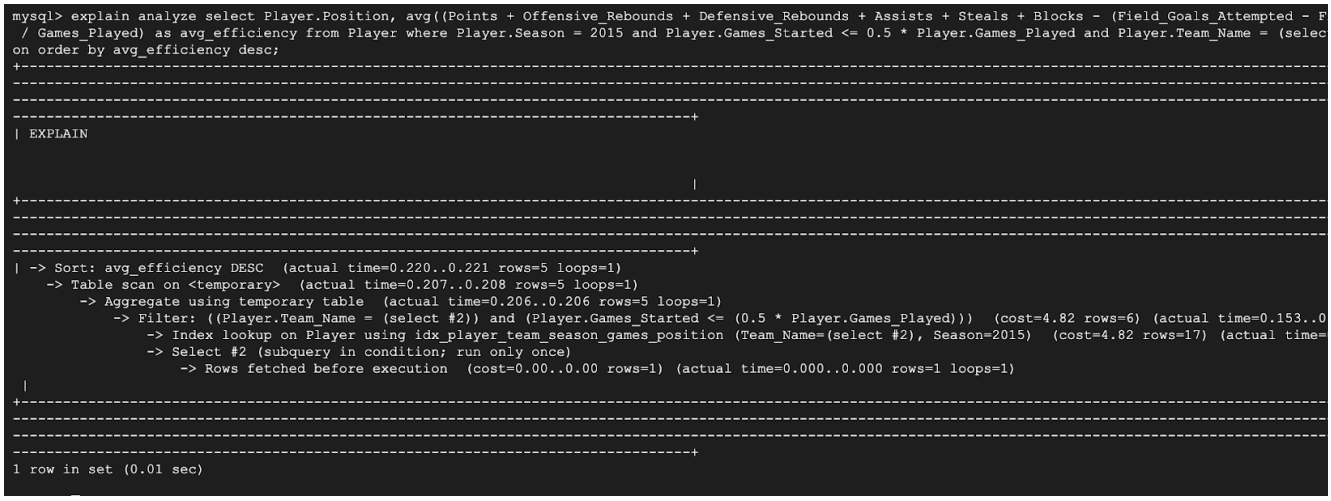
CREATE INDEX idx_user_favorite_team ON User(Favorite_Team);

```
mysql> explain analyze select Player.Position, avg((Points + Offensive_Rebounds + Defensive_Rebounds + Assists + Steals + Blocks - (Field_Goals_Attempted - Field_Goals_Made
 / Games_Played) as avg_efficiency from Player where Player.Season = 2015 and Player.Games_Started <= 0.5 * Player.Games_Played and Player.Team_Name = (select Favorite_Team
on order by avg_efficiency desc;
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------+
| EXPLAIN


                                              |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------+
| -> Sort: avg_efficiency DESC  (actual time=22.628..22.629 rows=5 loops=1)
    -> Table scan on <temporary>  (actual time=22.588..22.590 rows=5 loops=1)
        -> Aggregate using temporary table  (actual time=22.584..22.584 rows=5 loops=1)
            -> Filter: ((Player.Team_Name = (select #2)) and (Player.Season = 2015) and (Player.Games_Started <= (0.5 * Player.Games_Played)))  (cost=754.45 rows=24) (actua
                -> Table scan on Player  (cost=754.45 rows=7142) (actual time=0.083..19.882 rows=9455 loops=1)
                -> Select #2 (subquery in condition; run only once)
                    -> Rows fetched before execution  (cost=0.00..0.00 rows=1) (actual time=0.000..0.000 rows=1 loops=1)
 |
+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------------------+
1 row in set (0.03 sec)
```

After creating the index, it appears the cost of player.team_name = favorite_team is lowered from 1000.85 to 754.45.

The second thing we tried was a composite index on Player table for filtering by Team_Name, Season, Games_Started, and Position:

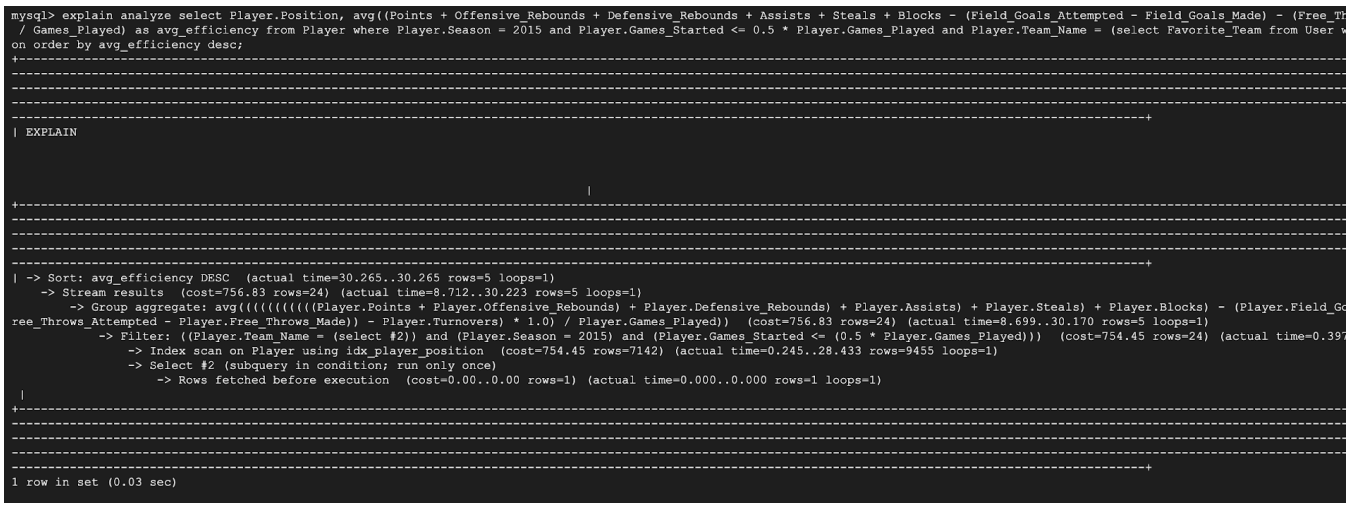
CREATE INDEX idx_player_team_season_games_position ON Player(Team_Name, Season, Games_Started, Position);



After implementing this, it appears that the cost is dramatically lowered from 1000.85 to 4.82 when filtering. This makes sense, since we are pre-computing every attribute that we need to extract. The index helps skip over non-relevant rows without scanning the full table.

The third thing we tried was an additional index on Position for grouping by position:

CREATE INDEX idx_player_position ON Player(Position);



Similarly to the first index we tried it appears that the cost is dramatically lowered from 1000.85 to 754.45 when filtering. This makes sense, since we are only pre-computing one value, this time it is player_position instead of favorite_team.

Looking at all three indexes, it appears that the second index is the preferred choice. This is because it is composite and pre-computes values for every attribute we need to extract, which makes it the obvious choice for the best index to use.