1. FINAL DDL

```
CREATE TABLE GAMES (
    game_date DATE NOT NULL,
    visitor_team_id VARCHAR(255) NOT NULL,
    visitor_pts INT,
    home_team_id VARCHAR(255) NOT NULL,
    home_pts INT,
    attendance INT,
    length_of_game INT,
    arena VARCHAR(50) NOT NULL,
    FOREIGN KEY (home_team_id) REFERENCES teams(team_name),
    FOREIGN KEY (visitor_team_id) REFERENCES teams(team_name)
);


CREATE TABLE PLAYERS (
    Rk INT,
    Player VARCHAR(255),
    Age INT,
    Team VARCHAR(10),
    Pos VARCHAR(10),
    G INT ,
    GS INT NOT NULL,
    MP DECIMAL(4,1) NOT NULL,
    FG DECIMAL(4,1) NOT NULL,
    FGA DECIMAL(4,1) NOT NULL,
    `FG%` DECIMAL(4,3) NOT NULL,
    `3P` DECIMAL(4,1) NOT NULL,
    `3PA` DECIMAL(4,1) NOT NULL,
    `3P%` DECIMAL(4,3) NOT NULL,
    `2P` DECIMAL(4,1) NOT NULL,
    `2PA` DECIMAL(4,1) NOT NULL,
    `2P%` DECIMAL(4,3) NOT NULL,
    `eFG%` DECIMAL(4,3) NOT NULL,
    FT DECIMAL(4,1) NOT NULL,
    FTA DECIMAL(4,1) NOT NULL,
    `FT%` DECIMAL(4,3) NOT NULL,
    ORB DECIMAL(4,1) NOT NULL,
    DRB DECIMAL(4,1) NOT NULL,
    TRB DECIMAL(4,1) NOT NULL,
    AST DECIMAL(4,1) NOT NULL,
    STL DECIMAL(4,1) NOT NULL,
    BLK DECIMAL(4,1) NOT NULL,
    TOV DECIMAL(4,1) NOT NULL,
    PF DECIMAL(4,1) NOT NULL,
    PTS DECIMAL(4,1) NOT NULL,
```

```
    Awards VARCHAR(50),
    Player_ID VARCHAR(50) NOT NULL,
    Year INT NOT NULL,
    FOREIGN KEY (Team) REFERENCES teams(team_id),
    PRIMARY KEY (Player_ID, Year)
);


CREATE TABLE game_stats (
    SEASON_YEAR VARCHAR(9),
    TEAM_ID INTEGER,
    TEAM_ABBREVIATION VARCHAR(5),
    TEAM_NAME VARCHAR(50),
    GAME_ID BIGINT,
    GAME_DATE DATE,
    MATCHUP VARCHAR(20),
    WL CHAR(1),
    MIN REAL,
    FGM INTEGER,
    FGA INTEGER,
    FG_PCT REAL,
    FG3M INTEGER,
    FG3A INTEGER,
    FG3_PCT REAL,
    FTM INTEGER,
    FTA INTEGER,
    FT_PCT REAL,
    OREB INTEGER,
    DREB INTEGER,
    REB INTEGER,
    AST INTEGER,
    TOV REAL,
    STL INTEGER,
    BLK INTEGER,
    BLKA INTEGER,
    PF INTEGER,
    PFD INTEGER,
    PTS INTEGER,
    PLUS_MINUS REAL,
    GP_RANK INTEGER,
    W_RANK INTEGER,
    L_RANK INTEGER,
    W_PCT_RANK INTEGER,
    MIN_RANK INTEGER,
    FGM_RANK INTEGER,
    FGA_RANK INTEGER,
    FG_PCT_RANK INTEGER,
```

```
    FG3M_RANK INTEGER,
    FG3A_RANK INTEGER,
    FG3_PCT_RANK INTEGER,
    FTM_RANK INTEGER,
    FTA_RANK INTEGER,
    FT_PCT_RANK INTEGER,
    OREB_RANK INTEGER,
    DREB_RANK INTEGER,
    REB_RANK INTEGER,
    AST_RANK INTEGER,
    TOV_RANK INTEGER,
    STL_RANK INTEGER,
    BLK_RANK INTEGER,
    BLKA_RANK INTEGER,
    PF_RANK INTEGER,
    PFD_RANK INTEGER,
    PTS_RANK INTEGER,
    PLUS_MINUS_RANK INTEGER,
    AVAILABLE_FLAG BOOLEAN,
    FOREIGN KEY (TEAM_ABBREVIATION) REFERENCES teams(team_id)
);

create table user_favorite (
    user_id int not null primary key,
    favorite_id varchar(50) not null,
    favorite_type boolean not null
);

create table user (
    user_id int not null unique primary key,
    username varchar(50) not null unique,
    email varchar(50) not null unique,
    password varchar(50) not null
);

create table teams (
    team_name varchar(255) unique,
    city varchar(255) not null,
    arena varchar(255) not null,
    team_id varchar(10) not null unique primary key
);
```

2. Screenshots of Evidence Showing Data Insertion + Four Advanced SQL Queries


```
mysql> CREATE TABLE game_stats (
    ->     SEASON_YEAR VARCHAR(9),
    ->     TEAM_ID INTEGER,
    ->     TEAM_ABBREVIATION VARCHAR(5),
    ->     TEAM_NAME VARCHAR(50),
    ->     GAME_ID BIGINT,
    ->     GAME_DATE DATE,
    ->     MATCHUP VARCHAR(20),
    ->     WL CHAR(1),
    ->     MIN REAL,
    ->     FGM INTEGER,
    ->     FGA INTEGER,
    ->     FG_PCT REAL,
    ->     FG3M INTEGER,
    ->     FG3A INTEGER,
    ->     FG3_PCT REAL,
    ->     FTM INTEGER,
    ->     FTA INTEGER,
    ->     FT_PCT REAL,
    ->     OREB INTEGER,
    ->     DREB INTEGER,
    ->     REB INTEGER,
    ->     AST INTEGER,
    ->     TOV REAL,
    ->     STL INTEGER,
    ->     BLK INTEGER,
    ->     BLKA INTEGER,
    ->     PF INTEGER,
    ->     PFD INTEGER,
    ->     PTS INTEGER,
    ->     PLUS_MINUS REAL,
    ->     GP_RANK INTEGER,
    ->     W_RANK INTEGER,
    ->     L_RANK INTEGER,
    ->     W_PCT_RANK INTEGER,
    ->     MIN_RANK INTEGER,
    ->     FGM_RANK INTEGER,
    ->     FGA_RANK INTEGER,
    ->     FG_PCT_RANK INTEGER,
    ->     FG3M_RANK INTEGER,
    ->     FG3A_RANK INTEGER,
    ->     FG3_PCT_RANK INTEGER,
    ->     FTM_RANK INTEGER,
    ->     FTA_RANK INTEGER,
    ->     FT_PCT_RANK INTEGER,
    ->     OREB_RANK INTEGER,
    ->     DREB_RANK INTEGER,
    ->     REB_RANK INTEGER,
    ->     AST_RANK INTEGER,
    ->     TOV_RANK INTEGER,
    ->     STL_RANK INTEGER,
    ->     BLK_RANK INTEGER,
    ->     BLKA_RANK INTEGER,
    ->     PF_RANK INTEGER,
    ->     PFD_RANK INTEGER,
    ->     PTS_RANK INTEGER,
    ->     PLUS_MINUS_RANK INTEGER,
    ->     AVAILABLE_FLAG BOOLEAN,
    ->     FOREIGN KEY (TEAM_ABBREVIATION) REFERENCES teams(team_id)
    -> );
Query OK, 0 rows affected (0.33 sec)
```

```
mysql> create table user_favorite (
    ->     user_id int not null primary key,
    ->     favorite_id varchar(50) not null,
    ->     favorite_type boolean not null
    -> );
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> create table user (
    ->     user_id int not null unique primary key,
    ->     username varchar(50) not null unique,
    ->     email varchar(50) not null unique,
    ->     password varchar(50) not null
    -> );
Query OK, 0 rows affected (0.24 sec)
```

```
mysql> CREATE TABLE PLAYERS (
    ->     Rk INT,
    ->     Player VARCHAR(255),
    ->     Age INT,
    ->     Team VARCHAR(10),
    ->     Pos VARCHAR(10),
    ->     G INT ,
    ->     GS INT NOT NULL,
    ->     MP DECIMAL(4,1) NOT NULL,
    ->     FG DECIMAL(4,1) NOT NULL,
    ->     FGA DECIMAL(4,1) NOT NULL,
    ->     `FG%` DECIMAL(4,3) NOT NULL,
    ->     `3P` DECIMAL(4,1) NOT NULL,
    ->     `3PA` DECIMAL(4,1) NOT NULL,
    ->     `3P%` DECIMAL(4,3) NOT NULL,
    ->     `2P` DECIMAL(4,1) NOT NULL,
    ->     `2PA` DECIMAL(4,1) NOT NULL,
    ->     `2P%` DECIMAL(4,3) NOT NULL,
    ->     `eFG%` DECIMAL(4,3) NOT NULL,
    ->     FT DECIMAL(4,1) NOT NULL,
    ->     FTA DECIMAL(4,1) NOT NULL,
    ->     `FT%` DECIMAL(4,3) NOT NULL,
    ->     ORB DECIMAL(4,1) NOT NULL,
    ->     DRB DECIMAL(4,1) NOT NULL,
    ->     TRB DECIMAL(4,1) NOT NULL,
    ->     AST DECIMAL(4,1) NOT NULL,
    ->     STL DECIMAL(4,1) NOT NULL,
    ->     BLK DECIMAL(4,1) NOT NULL,
    ->     TOV DECIMAL(4,1) NOT NULL,
    ->     PF DECIMAL(4,1) NOT NULL,
    ->     PTS DECIMAL(4,1) NOT NULL,
    ->     Awards VARCHAR(50),
    ->     Player_ID VARCHAR(50) NOT NULL,
    ->     Year INT NOT NULL,
    ->     FOREIGN KEY (Team) REFERENCES teams(team_id),
    ->     PRIMARY KEY (Player_ID, Year)
    -> );
Query OK, 0 rows affected (0.19 sec)
```

```
mysql> CREATE TABLE GAMES (
    ->     game_date DATE NOT NULL,
    ->     visitor_team_id VARCHAR(255) NOT NULL,
    ->     visitor_pts INT,
    ->     home_team_id VARCHAR(255) NOT NULL,
    ->     home_pts INT,
    ->     attendance INT,
    ->     length_of_game INT,
    ->     arena VARCHAR(50) NOT NULL,
    ->     FOREIGN KEY (home_team_id) REFERENCES teams(team_name),
    ->     FOREIGN KEY (visitor_team_id) REFERENCES teams(team_name)
    -> );
Query OK, 0 rows affected (0.14 sec)
```

```
mysql> create table user_favorite (
    ->     user_id int not null primary key,
    ->     favorite_id varchar(50) not null,
    ->     favorite_type boolean not null
    -> );
Query OK, 0 rows affected (0.08 sec)
```

```
mysql> SELECT t.team_name, SUM(gs.pts) AS total_pts, SUM(gs.reb) AS total_rebounds
    -> FROM game_stats AS gs
    -> JOIN teams AS t ON gs.team_abbreviation = t.team_id
    -> GROUP BY t.team_name
    -> ORDER BY total_pts DESC;
+------------------------+-----------+----------------+
| team_name              | total_pts | total_rebounds |
+------------------------+-----------+----------------+
| Golden State Warriors  |    122280 |          48866 |
| Houston Rockets        |    121715 |          48559 |
| Denver Nuggets         |    121441 |          49495 |
| Oklahoma City Thunder  |    120802 |          50119 |
| Los Angeles Clippers   |    120547 |          48224 |
| San Antonio Spurs      |    119434 |          48528 |
| Phoenix Suns           |    119234 |          48050 |
| Milwaukee Bucks        |    119192 |          49323 |
| Sacramento Kings       |    118693 |          47754 |
| Los Angeles Lakers     |    118643 |          49331 |
| Portland Trail Blazers |    118624 |          48704 |
| Dallas Mavericks       |    118554 |          47141 |
| Minnesota Timberwolves |    118231 |          47630 |
| Toronto Raptors        |    118203 |          47693 |
| Atlanta Hawks          |    117980 |          47288 |
| Washington Wizards     |    117863 |          47514 |
| Indiana Pacers         |    117804 |          48199 |
| Utah Jazz              |    117124 |          49018 |
| Philadelphia 76ers     |    116599 |          48294 |
| Miami Heat             |    116252 |          46768 |
| Cleveland Cavaliers    |    115356 |          47285 |
| Memphis Grizzlies      |    115192 |          48367 |
| New York Knicks        |    114952 |          48264 |
| Chicago Bulls          |    114395 |          48878 |
| Orlando Magic          |    113825 |          48122 |
| Charlotte Hornets      |    113611 |          47321 |
| Detroit Pistons        |    113451 |          47884 |
| Brooklyn Nets          |    102956 |          41871 |
| New Orleans Pelicans   |     96298 |          39227 |
+------------------------+-----------+----------------+
```

```
mysql> SELECT g.attendance, g.home_team_id
    -> FROM GAMES AS g
    -> WHERE g.attendance >(SELECT AVG(attendance) FROM GAMES)
    -> ORDER BY g.attendance DESC
    -> LIMIT 15;
+------------+-----------------+
| attendance | home_team_id    |
+------------+-----------------+
|      22491 | Chicago Bulls   |
|      22062 | Detroit Pistons |
|      21957 | Chicago Bulls   |
|      21647 | Chicago Bulls   |
|      21511 | Chicago Bulls   |
|      21391 | Chicago Bulls   |
|      21297 | Chicago Bulls   |
|      21234 | Chicago Bulls   |
|      21116 | Chicago Bulls   |
|      21045 | Chicago Bulls   |
|      20943 | Chicago Bulls   |
|      20938 | Chicago Bulls   |
|      20923 | Chicago Bulls   |
|      20697 | Chicago Bulls   |
|      20667 | Chicago Bulls   |
+------------+-----------------+
15 rows in set (0.01 sec)
```

Go Bulls!

```
mysql> SELECT p.Player, p.Year, p.PTS, t.team_name
    -> FROM PLAYER AS p
    -> JOIN teams as t ON p.Team = t.team_id
    -> WHERE p.PTS  > (
    ->     SELECT AVG(p2.PTS)
    ->     FROM PLAYER p2
    ->     WHERE p2.Year = p.Year
    -> )
    -> LIMIT 15;
+----------------------+------+------+-----------------------+
| Player               | Year | PTS  | team_name             |
+----------------------+------+------+-----------------------+
| Bam Adebayo          | 2024 | 19.3 | Miami Heat            |
| Santi Aldama         | 2024 | 10.7 | Memphis Grizzlies     |
| Jarrett Allen        | 2024 | 16.5 | Cleveland Cavaliers   |
| Giannis Antetokounmpo | 2024 | 30.4 | Milwaukee Bucks       |
| Cole Anthony         | 2024 | 11.6 | Orlando Magic         |
| Deni Avdija          | 2024 | 14.7 | Washington Wizards    |
| Deandre Ayton        | 2024 | 16.7 | Portland Trail Blazers |
| Lonzo Ball           | 2025 |  7.6 | Chicago Bulls         |
| Paolo Banchero       | 2024 | 22.6 | Orlando Magic         |
| Desmond Bane         | 2024 | 23.7 | Memphis Grizzlies     |
| Harrison Barnes      | 2024 | 12.2 | Sacramento Kings      |
| Scottie Barnes       | 2024 | 19.9 | Toronto Raptors       |
| Jamison Battle       | 2025 |  6.5 | Toronto Raptors       |
| Malik Beasley        | 2024 | 11.3 | Milwaukee Bucks       |
| Saddiq Bey           | 2024 | 13.7 | Atlanta Hawks         |
+----------------------+------+------+-----------------------+
15 rows in set (0.02 sec)
```

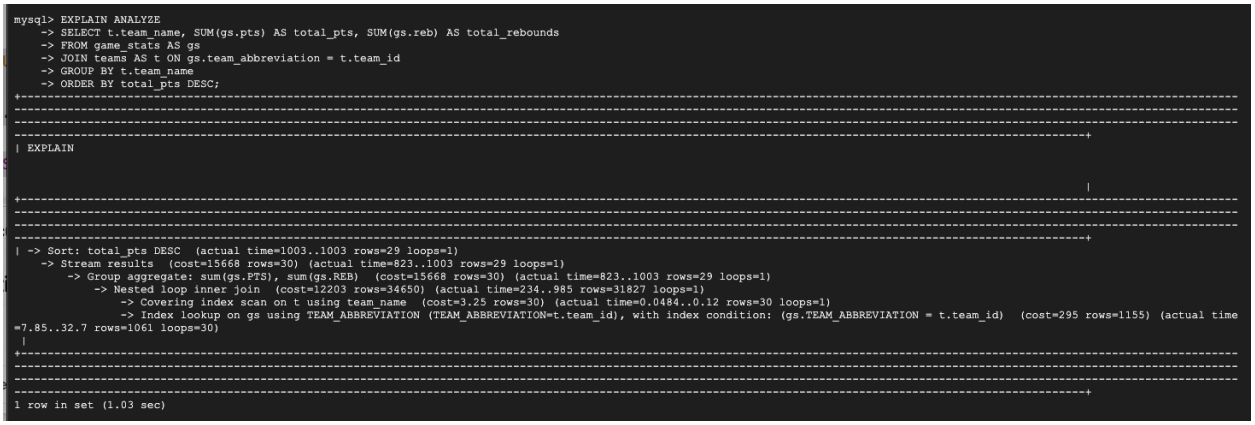
```
Database changed
mysql> SELECT
    ->     p.Player,
    ->     SUM(p.PTS + p.AST + p.TRB) AS total_points,
    ->     p.G AS min_games_required
    -> FROM
    ->     PLAYER AS p
    -> WHERE
    ->     p.G >= 65
    -> GROUP BY
    ->     p.Player, p.G
    -> ORDER BY
    ->     total_points DESC, min_games_required DESC
    -> LIMIT 15;
+-----------------------+--------------+--------------------+
| Player                | total_points | min_games_required |
+-----------------------+--------------+--------------------+
| Luka Dončić           |         52.9 |                 70 |
| Giannis Antetokounmpo |         48.4 |                 73 |
| Nikola Jokić          |         47.8 |                 79 |
| Shai Gilgeous-Alexander |         41.8 |                 75 |
| Domantas Sabonis      |         41.3 |                 82 |
| LeBron James          |         41.3 |                 71 |
| Anthony Davis         |         40.8 |                 76 |
| Jayson Tatum          |         39.9 |                 74 |
| Jalen Brunson         |         39.0 |                 77 |
| Kevin Durant          |         38.7 |                 75 |
| Devin Booker          |         38.5 |                 68 |
| De'Aaron Fox          |         36.8 |                 74 |
| Anthony Edwards       |         36.4 |                 79 |
| Stephen Curry         |         36.0 |                 74 |
| Victor Wembanyama     |         35.9 |                 71 |
+-----------------------+--------------+--------------------+
15 rows in set (0.00 sec)
```

3. Index Analysis Details (Performance Gains/Degradations for Each Indexing Configuration)

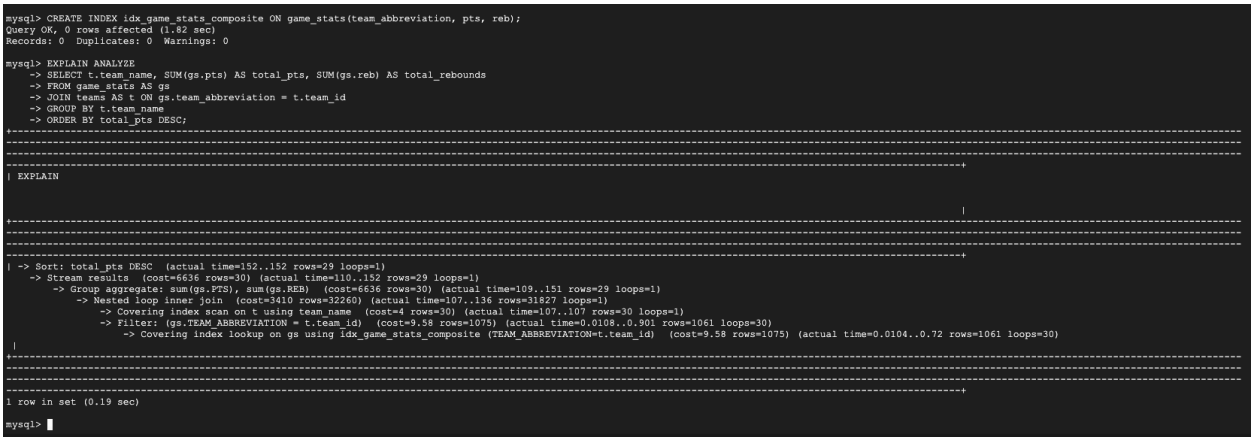
QUERY 1

The composite index on game_stats(team_abbrevation, pts,reb) is created for the purpose of optimizing the JOIN condition and the aggregation operations on pts and reb. By combining columns in a single index, composite index like this would reduce the overall cost and execution time.

Before Index

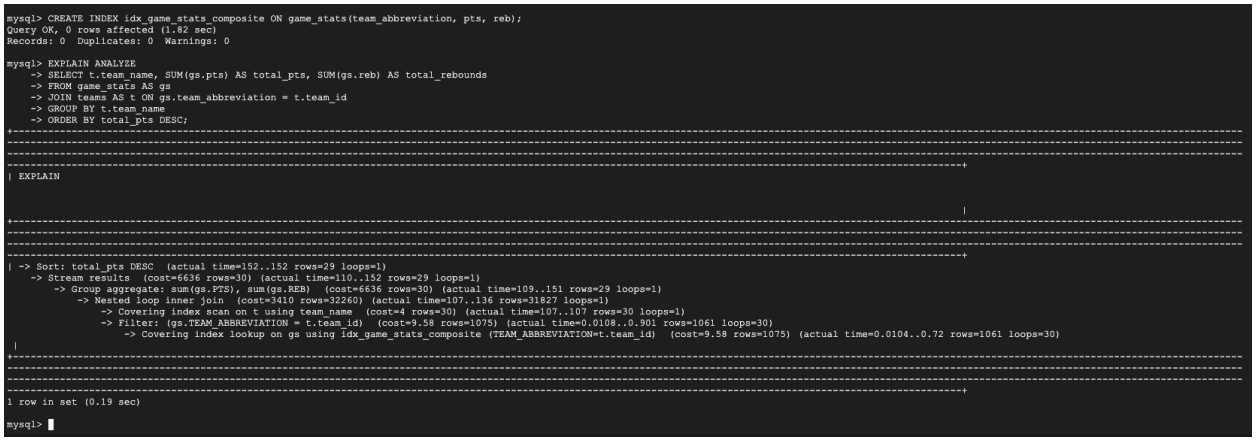


After Index1:
CREATE INDEX idx_game_stats_combination ON game_stats(team_abbreviation, pts,reb);

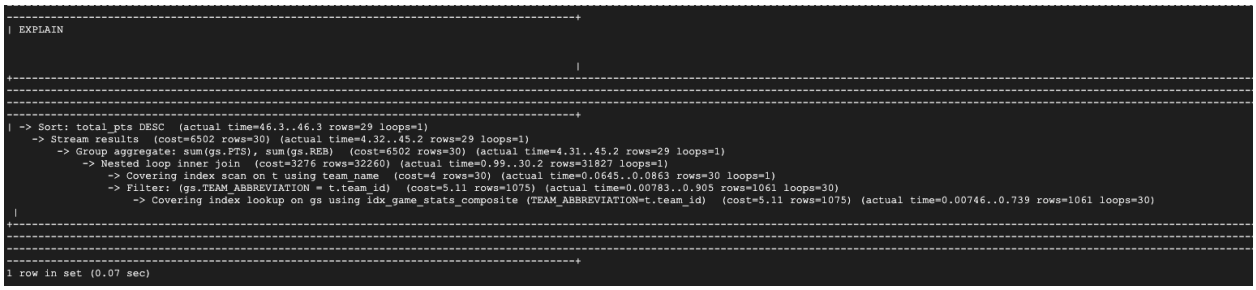


After Index2:

CREATE INDEX idx_game_stats_team_abbreviation ON game_stats(team_abbreviation);



After Index3:
CREATE INDEX idx_game_stats_pts_reb ON game_stats(pts, reb);



After indexing, the overall cost dropped from 15,668 to 6,636 and the actual execution time decreased from about 1.03 seconds to 0.19 seconds, which implies that the composite indexing method we applied here was able to enhance the query's performance. From all 3 indices, it seems that the third one has the lowest cost for the filter, with a cost of only 5.11 compared to the others.

QUERY 2

Index on GAMES(attendance) is used to support WHERE clause the ORDER BY operation. Index like this can decrease the number of rows scanned, while enhancing the query's execution time.

Before Index:

```
mysql> EXPLAIN ANALYZE
    -> SELECT g.attendance, g.home_team_id
    -> FROM GAMES AS g
    -> WHERE g.attendance > (SELECT AVG(attendance) FROM GAMES)
    -> ORDER BY g.attendance DESC
    -> LIMIT 15;
+--------------------------------------------------------------------------------+
| EXPLAIN                                                                        |
+--------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=39.8 rows=15) (actual time=3.59..3.59 rows=15 loops=1)
    -> Sort: g.attendance DESC, limit input to 15 row(s) per chunk  (cost=39.8 rows=1126) (actual time=3.59..3.59 rows=15 loops=1)
        -> Filter: (g.attendance > (select #2))  (cost=39.8 rows=1126) (actual time=2.58..3.37 rows=576 loops=1)
            -> Table scan on g  (cost=39.8 rows=1126) (actual time=0.0783..0.723 rows=1126 loops=1)
            -> Select #2 (subquery in condition; run only once)
                -> Aggregate: avg(GAMES.attendance)  (cost=227 rows=1) (actual time=1.74..1.74 rows=1 loops=1)
                    -> Table scan on GAMES  (cost=115 rows=1126) (actual time=0.00596..0.72 rows=1126 loops=1)
 |
+--------------------------------------------------------------------------------+
1 row in set (0.03 sec)

mysql>
```

After Index1:

CREATE INDEX idx_games_attendence ON GAMES(attendance);

```
mysql> CREATE INDEX idx_games_attendance ON GAMES(attendance);
Query OK, 0 rows affected (0.44 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN ANALYZE
    -> SELECT g.attendance, g.home_team_id
    -> FROM GAMES AS g
    -> WHERE g.attendance > (SELECT AVG(attendance) FROM GAMES)
    -> ORDER BY g.attendance DESC
    -> LIMIT 15;
+--------------------------------------------------------------------------------+
| EXPLAIN                                                                        |
+--------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=115 rows=15) (actual time=1.2..1.26 rows=15 loops=1)
    -> Filter: (g.attendance > (select #2))  (cost=115 rows=576) (actual time=1.2..1.26 rows=15 loops=1)
        -> Index range scan on g using idx_games_attendance over (18120 < attendance) (reverse)  (cost=115 rows=576) (actual time=1.19..1.25 rows=15 loops=1)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(GAMES.attendance)  (cost=227 rows=1) (actual time=1.12..1.12 rows=1 loops=1)
                -> Covering index scan on GAMES using idx_games_attendance  (cost=115 rows=1126) (actual time=0.0398..0.906 rows=1126 loops=1)
 |
+--------------------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql>
```

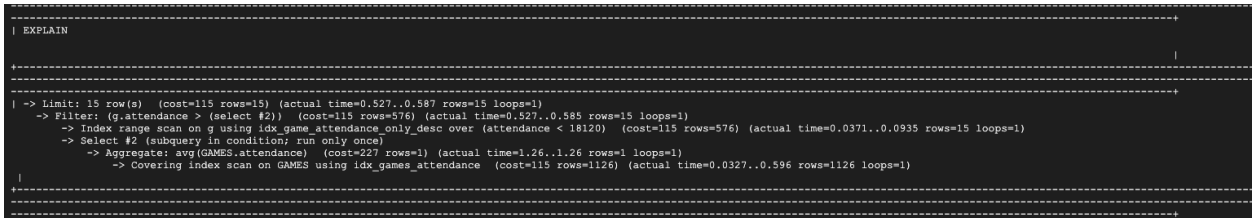
After Index2:

CREATE INDEX idx_game_attendance_game_date ON GAMES(attendance, game_date);

```
---------------------------------------------------------------------------------+
| EXPLAIN                                                                        |
+--------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (cost=115 rows=15) (actual time=1.13..1.19 rows=15 loops=1)
    -> Filter: (g.attendance > (select #2))  (cost=115 rows=576) (actual time=1.13..1.19 rows=15 loops=1)
        -> Index range scan on g using idx_games_attendance over (18120 < attendance) (reverse)  (cost=115 rows=576) (actual time=0.63..0.683 rows=15 loops=1)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(GAMES.attendance)  (cost=227 rows=1) (actual time=37.3..37.3 rows=1 loops=1)
                -> Covering index scan on GAMES using idx_games_attendance  (cost=115 rows=1126) (actual time=29..36.6 rows=1126 loops=1)
 |
+--------------------------------------------------------------------------------+
```

After Index3:

CREATE INDEX idx_game_attendance_only_desc ON GAMES(attendance DESC);

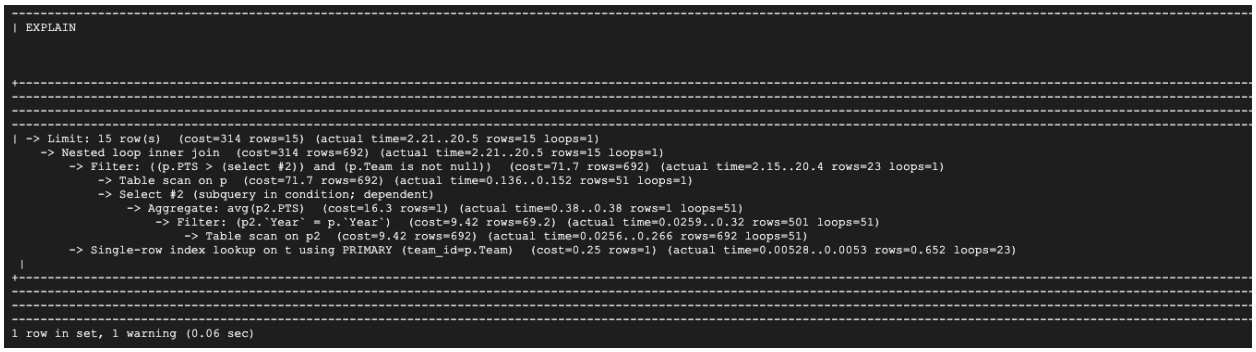


After indexing, the actual execution time dropped from about 3.59 seconds to roughly 1.2 seconds, and the number of rows processed in the filter step was decreased from 1126 to 576. This suggests that adding indexes on the attendance column has enhanced the query's performance. Comparing all 3 indices, it seems that they are all functionally similar with the same costs, and only the time changing, which is not a strong indicator of performance.

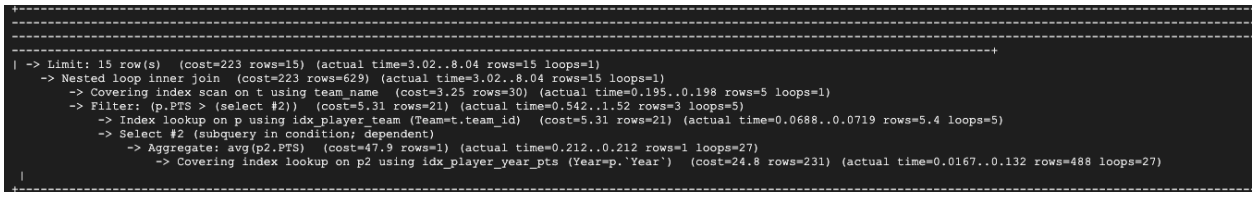
QUERY 3

The join columns are indexed via establishing an index on PLAYER(Team) and on teams(team_id). The composite index on PLAYER (Year, Pts) is aim to help the subquery filtering.

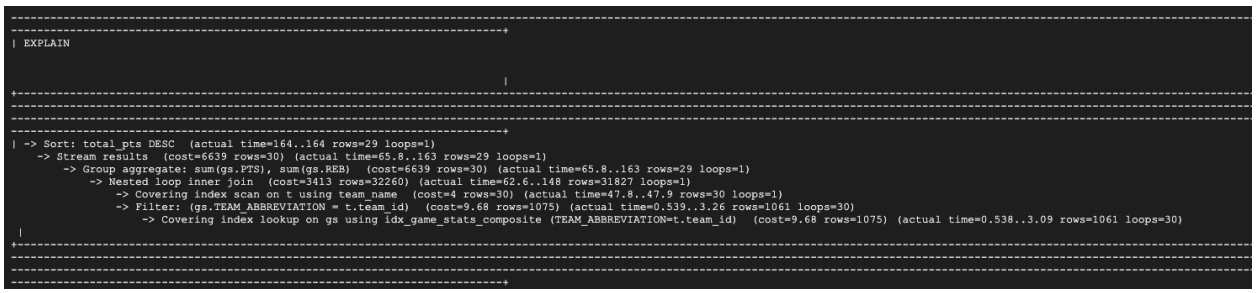
Before index:



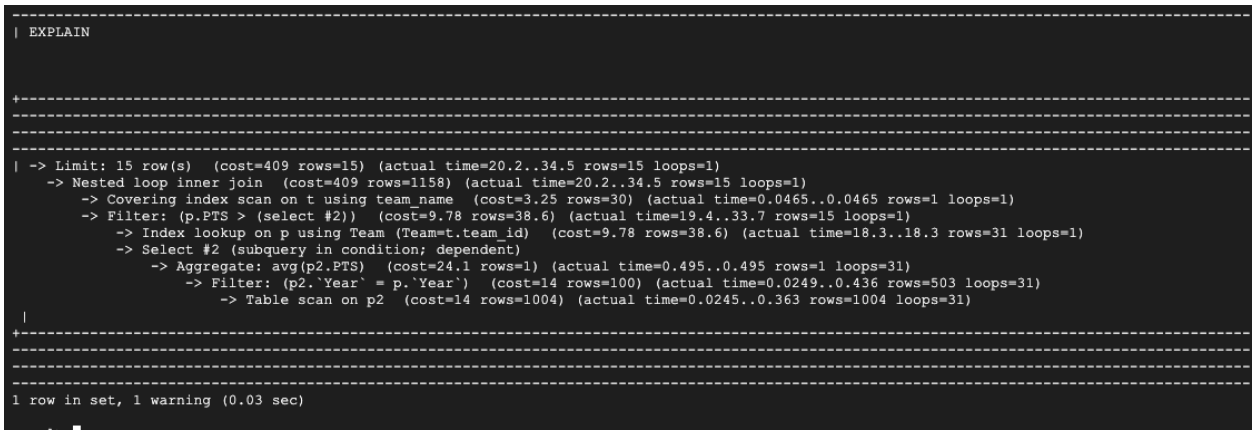
After index1:
CREATE INDEX idx_player_team ON PLAYER(Team);



After index2:
CREATE INDEX idx_player_year_pts ON PLAYER(Year, PTS);



After index3:
CREATE INDEX idx_teams_team_id ON teams(team_id);



After indexing, the number of rows that were processed went from 314 to 223 and the actual time decreased to 3.02..8.04 from 2.21..20.5, suggesting that this method of indexing improved the performance. Comparing all 3 of the indices, it seems that the second one has the best performance as it has the lowest cost by far out of all of them, which is a strong indicator of increased performance.

Query 4:
Before Index:

```
| EXPLAIN
                                                                                          |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=1.17..1.17 rows=15 loops=1)
    -> Sort: total_points DESC, p.G DESC, limit input to 15 row(s) per chunk  (actual time=1.16..1.17 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=0.912..0.966 rows=199 loops=1)
            -> Aggregate using temporary table  (actual time=0.911..0.911 rows=199 loops=1)
                -> Filter: (p.G >= 65)  (cost=71.7 rows=231) (actual time=0.173..0.612 rows=199 loops=1)
                    -> Table scan on p  (cost=71.7 rows=692) (actual time=0.167..0.553 rows=692 loops=1)
 |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------+
1 row in set (0.03 sec)
```

After Index:

CREATE INDEX idx_player_pts ON PLAYER(PTS);

```
------------------------------------------------------------------------------------------+
| EXPLAIN
                                                                                          |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=1.87..1.87 rows=15 loops=1)
    -> Sort: total_points DESC, p.G DESC, limit input to 15 row(s) per chunk  (actual time=1.87..1.87 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=1.73..1.76 rows=199 loops=1)
            -> Aggregate using temporary table  (actual time=1.73..1.73 rows=199 loops=1)
                -> Filter: (p.G >= 65)  (cost=71.7 rows=231) (actual time=0.104..0.545 rows=199 loops=1)
                    -> Table scan on p  (cost=71.7 rows=692) (actual time=0.1..0.488 rows=692 loops=1)
 |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------+
1 row in set (0.03 sec)
```

After Index2:

CREATE INDEX idx_player_filter_group ON PLAYER (G, Player);

```
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------------------------------------------------+
| EXPLAIN
                                                          |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=4.44..4.44 rows=15 loops=1)
    -> Sort: total_points DESC, p.G DESC, limit input to 15 row(s) per chunk  (actual time=4.44..4.44 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=3.35..3.39 rows=244 loops=1)
            -> Aggregate using temporary table  (actual time=3.35..3.35 rows=244 loops=1)
                -> Index range scan on p using idx_player_filter_group over (65 <= G), with index condition: (p.G >= 65)  (cost=113 rows=250) (actual time=0.496..1.77 rows=250 loops=1)
 |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------------------------------------------------+
1 row in set (0.03 sec)
```

After Index3:

CREATE INDEX idx_player_sorting ON PLAYERS (PTS, AST, TRB, G);

```
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
-----------------------------------------------------+
| EXPLAIN
                                                     |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
-----------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=1.12..1.13 rows=15 loops=1)
    -> Sort: total_points DESC, p.G DESC, limit input to 15 row(s) per chunk  (actual time=1.12..1.12 rows=15 loops=1)
        -> Table scan on <temporary>  (actual time=1.01..1.04 rows=244 loops=1)
            -> Aggregate using temporary table  (actual time=1..1 rows=244 loops=1)
                -> Index range scan on p using idx_player_filter_group over (65 <= G), with index condition: (p.G >= 65)  (cost=113 rows=250) (actual time=0.0378..0.613 rows=250 loops=1)
 |
+------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
-----------------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

The overall query execution seems to have slowed down by around 0.6 but the costs are the same, which suggests that there is no difference between the two results. From the 3 different indices, it seems that the first one has the best performance as it has the lowest cost, and the other two both increased the cost, which suggests that the first one has the best performance.

4. Justification for Final Indexing Design

Our result from Query 4 suggests that indexing a column not directly involved with GROUP BY, JOIN, or WHERE operations might not drive the performance improvements. Specifically, in this case, it might otherwise degrade performance slightly. This drives us not to implement an index on PLAYER (PTS) for Q4. On the other hand, the composite and targeted indexes utilized from Q1-Q3 have shown significant performance gains. The overall cost, execution time, and the rows processed have reduced. This suggests the effectiveness in optimizing database operations using this indexing strategies. As a result, the final index design incorporates the indexes from Q1-3.