

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

```

-> );
Query OK, 0 rows affected (0.19 sec)

mysql> select count(*) from GAMES;
+-----+
| count(*) |
+-----+
|      1126 |
+-----+
1 row in set (0.02 sec)

mysql>

```

✓ Imported from per_game_2024_2025.csv to cs411-primarykeyplayers	01:05:42 GMT-5
✓ Imported from nba_GAMES.csv to cs411-primarykeyplayers	01:05:12 GMT-5
✓ Imported from nba_GAMES.csv to cs411-primarykeyplayers	00:44:21 GMT-5

```

-> );
Query OK, 0 rows affected (0.11 sec)

mysql> select count(*) from game_stats;
+-----+
| count(*) |
+-----+
|      2127 |
+-----+
1 row in set (0.01 sec)

mysql>

```

1 file successfully uploaded X

✓ Imported from nba_2010_to_2024_reg_season_totals.csv to cs411-primarykeyplayers	23:22:35 GMT-5
✓ Imported from nba_2010_to_2024_reg_season_totals.csv	Complete
✓ Imported from	23:17:24 GMT-5

```

1 row in set (0.02 sec)

mysql> select count(*) from PLAYERS;
+-----+
| count(*) |
+-----+
|      1004 |
+-----+
1 row in set (0.03 sec)

mysql>

```

✓ Imported from per_game_2024_2025.csv to cs411-primarykeyplayers	01:05:42 GMT-5
✓ Imported from nba_GAMES.csv to cs411-primarykeyplayers	01:05:12 GMT-5
✓ Imported from nba_GAMES.csv to cs411-primarykeyplayers	00:44:21 GMT-5

```

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 Avdiya	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)

QUERY 1

Before Index

After Index1:

```
mysql> CREATE INDEX idx_game_stats_composite ON game_stats(team_abbreviation, pts, reb);
Query OK, 0 rows affected (1.82 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN ANALYZE
-> SELECT t.team_name, SUM(gs.pst) 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;

+-----+
| EXPLAIN |
+-----+
|         |
+-----+
|         |
+-----+
| -> Sort: total_pts DESC (actual time=152..152 rows=29 loops=1) |
+-----+
|   -> Stream results (cost=6636 rows=30) (actual time=110..152 rows=29 loops=1) |
+-----+
|     -> Group aggregate: sum(gs.PTS), sum(gs.REB) (cost=6636 rows=30) (actual time=109..151 rows=29 loops=1) |
+-----+
|       -> Nested loop inner join (cost=8410 rows=3240) (actual time=107..136 rows=3127 loops=1) |
+-----+
|         -> Covering index scan on t using team_name (cost=4 rows=30) (actual time=107..107 rows=30 loops=1) |
+-----+
|           -> Filter: (gs.TEAM_ABBREVIATION = t.team_id) (cost=9.58 rows=1075) (actual time=0.0108..0.901 rows=1061 loops=30) |
+-----+
|             -> Covering index lookup on gs using idx_game_stats_composite (TEAM_ABBREVIATION=t.team_id) (cost=9.58 rows=1075) (actual time=0.0104..0.72 rows=1061 loops=30) |
+-----+
|         |
+-----+
| 1 row in set (0.19 sec) |
+-----+

mysql>
```

9

```
CREATE INDEX idx_game_stats_team_abbreviation ON
game_stats(team_abbreviation);
```

```
mysql> CREATE INDEX idx_game_stats_composite ON game_stats(team_abbreviation, pts, reb);
Query OK, 0 rows affected (1.82 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> EXPLAIN ANALYZE
-> 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;
+-----+
| EXPLAIN |
+-----+
+-----+
| -> Sort: total_pts DESC (actual time=152.152 rows=29 loops=1) |
| -> Stream results (cost=6636 rows=30) (actual time=119.152 rows=29 loops=1) |
| -> Group aggregate: sum(gs.pts), sum(gs.reb) (cost=6636 rows=30) (actual time=109.151 rows=29 loops=1) |
| -> Nested loop inner join (cost=3410 rows=32260) (actual time=107.136 rows=31827 loops=1) |
| -> Covering index scan on t using team_name (cost=4 rows=30) (actual time=107.107 rows=30 loops=1) |
| -> Filter: (gs.TEAM ABBREVIATION = t.team_id) (cost=9.58 rows=1075) (actual time=0.0108.0.901 rows=1061 loops=30) |
| -> Covering index lookup on gs using idx_game_stats_composite (TEAM ABBREVIATION=t.team_id) (cost=9.58 rows=1075) (actual time=0.0104.0.72 rows=1061 loops=30) |
+-----+
1 row in set (0.19 sec)

mysql>
```

After Index3:

```
CREATE INDEX idx_game_stats_pts_reb ON game_stats(pts, reb);
```

```
+-----+
| EXPLAIN |
+-----+
+-----+
| -> Sort: total_pts DESC (actual time=46.3.46.3 rows=29 loops=1) |
| -> Stream results (cost=6502 rows=30) (actual time=4.32.45.2 rows=29 loops=1) |
| -> Group aggregate: sum(gs.pts), sum(gs.reb) (cost=6502 rows=30) (actual time=4.31.45.2 rows=29 loops=1) |
| -> Nested loop inner join (cost=3276 rows=32260) (actual time=0.99.30.2 rows=31827 loops=1) |
| -> Covering index scan on t using team_name (cost=4 rows=30) (actual time=0.0645.0.0863 rows=30 loops=1) |
| -> Filter: (gs.TEAM ABBREVIATION = t.team_id) (cost=5.11 rows=1075) (actual time=0.00783.0.905 rows=1061 loops=30) |
| -> Covering index lookup on gs using idx_game_stats_composite (TEAM ABBREVIATION=t.team_id) (cost=5.11 rows=1075) (actual time=0.00746.0.739 rows=1061 loops=30) |
+-----+
1 row in set (0.07 sec)
```

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=10.4 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_attendance 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)
+-----+
|
+-----+
1 row in set (0.03 sec)
```

After Index3:

CREATE INDEX idx_game_attendance_only_desc ON GAMES(attendance DESC);

```

| EXPLAIN
|
+-----+
|
+-----+
| -> Limit: 15 row(s) (cost=115 rows=15) (actual time=0.527..0.587 rows=15 loops=1)
|   -> Filter: (g.attendance > (select #2)) (cost=115 rows=576) (actual time=0.527..0.585 rows=15 loops=1)
|     -> Index range scan on g using idx_game_attendance_only_desc over (attendance < 18120) (cost=115 rows=576) (actual time=0.0371..0.0935 rows=15 loops=1)
|       -> Select #2 (subquery in condition; run only once)
|         -> Aggregate: avg(GAMES.attendance) (cost=227 rows=1) (actual time=1.26..1.26 rows=1 loops=1)
|           -> Covering index scan on GAMES using idx_games_attendance (cost=115 rows=1126) (actual time=0.0327..0.596 rows=1126 loops=1)
|
+-----+

```

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:

```

| EXPLAIN
|
+-----+
|
+-----+
| -> Limit: 15 row(s) (cost=314 rows=15) (actual time=2.21..20.5 rows=15 loops=1)
|   -> Nested loop inner join (cost=314 rows=692) (actual time=2.21..20.5 rows=15 loops=1)
|     -> Filter: ((p.PTS > (select #2)) and (p.Team is not null)) (cost=71.7 rows=692) (actual time=2.15..20.4 rows=23 loops=1)
|       -> Table scan on p (cost=71.7 rows=692) (actual time=0.136..0.152 rows=51 loops=1)
|         -> Select #2 (subquery in condition; dependent)
|           -> Aggregate: avg(p2.PTS) (cost=16.3 rows=1) (actual time=0.38..0.38 rows=1 loops=51)
|             -> Filter: (p2.'Year' = p.'Year') (cost=9.42 rows=69.2) (actual time=0.0259..0.32 rows=501 loops=51)
|               -> Table scan on p2 (cost=9.42 rows=692) (actual time=0.0256..0.266 rows=692 loops=51)
|                 -> Single-row index lookup on t using PRIMARY (team_id=p.Team) (cost=0.25 rows=1) (actual time=0.00528..0.0053 rows=0.652 loops=23)
|
+-----+
|
+-----+
| 1 row in set, 1 warning (0.06 sec)

```

After index1:

CREATE INDEX idx_player_team ON PLAYER(Team);

```

+-----+
|
+-----+
| -> Limit: 15 row(s) (cost=223 rows=15) (actual time=3.02..8.04 rows=15 loops=1)
|   -> Nested loop inner join (cost=223 rows=629) (actual time=3.02..8.04 rows=15 loops=1)
|     -> Covering index scan on t using team name (cost=3.25 rows=30) (actual time=0.195..0.198 rows=5 loops=1)
|       -> Filter: (p.PTS > (select #2)) (cost=5.31 rows=21) (actual time=0.542..1.52 rows=3 loops=5)
|         -> Index lookup on p using idx_player_team (Team=t.team_id) (cost=5.31 rows=21) (actual time=0.0688..0.0719 rows=5.4 loops=5)
|           -> Select #2 (subquery in condition; dependent)
|             -> Aggregate: avg(p2.PTS) (cost=47.9 rows=1) (actual time=0.212..0.212 rows=1 loops=27)
|               -> Covering index lookup on p2 using idx_player_year_pts (Year=p.'Year') (cost=24.8 rows=231) (actual time=0.0167..0.132 rows=488 loops=27)
|
+-----+

```

After index2:

```
CREATE INDEX idx_player_year_pts ON PLAYER(Year, PTS);
```

```
-----+-----
| EXPLAIN
|
+-----+-----+
|
+-----+-----+
| -> Sort: total_pts DESC (actual time=164..164 rows=29 loops=1)
|   -> Stream results (cost=6639 rows=30) (actual time=65.8..163 rows=29 loops=1)
|     -> Group aggregate: sum(gs.PTS), sum(gs.REB) (cost=6639 rows=30) (actual time=65.8..163 rows=29 loops=1)
|       -> Nested loop inner join (cost=3413 rows=32260) (actual time=62.6..148 rows=31827 loops=1)
|         -> Covering index scan on t using team_name (cost=4 rows=30) (actual time=47.8..47.9 rows=30 loops=1)
|           -> Filter: (gs.TEAM ABBREVIATION = t.team_id) (cost=9.68 rows=1075) (actual time=0.539..3.26 rows=1061 loops=30)
|             -> Covering index lookup on gs using idx_game_stats_composite (TEAM ABBREVIATION=t.team_id) (cost=9.68 rows=1075) (actual time=0.538..3.09 rows=1061 loops=30)
|
+-----+-----+
|
+-----+-----+
|
```

After index3:

```
CREATE INDEX idx_teams_team_id ON teams(team_id);
```

```
-----+-----
| EXPLAIN
|
+-----+-----+
|
+-----+-----+
| -> Limit: 15 row(s) (cost=409 rows=15) (actual time=20.2..34.5 rows=15 loops=1)
|   -> Nested loop inner join (cost=409 rows=1158) (actual time=20.2..34.5 rows=15 loops=1)
|     -> Covering index scan on t using team_name (cost=3.25 rows=30) (actual time=0.0465..0.0465 rows=1 loops=1)
|     -> Filter: (p.PTS > (select #2)) (cost=9.78 rows=38.6) (actual time=19.4..33.7 rows=15 loops=1)
|       -> Index lookup on p using Team (Team=t.team_id) (cost=9.78 rows=38.6) (actual time=18.3..18.3 rows=31 loops=1)
|         -> Select #2 (subquery in condition; dependent)
|           -> Aggregate: avg(p2.PTS) (cost=24.1 rows=1) (actual time=0.495..0.495 rows=1 loops=31)
|             -> Filter: (p2.'Year' = p.'Year') (cost=14 rows=100) (actual time=0.0249..0.436 rows=503 loops=31)
|               -> Table scan on p2 (cost=14 rows=1004) (actual time=0.0245..0.363 rows=1004 loops=31)
|
+-----+-----+
|
+-----+-----+
|
+-----+-----+
| 1 row in set, 1 warning (0.03 sec)
|
```

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.