Stage 3

Pt 1 implementation:

1. **5 Tables**
2. **DDL Commands used to create each table**

```
CREATE DATABASE FantasyResearchAssistant;

USE FantasyResearchAssistant;

CREATE TABLE UserInfo (
   user_id INT PRIMARY KEY,
   username VARCHAR(100) NOT NULL,
   email VARCHAR(100) NOT NULL,
   password VARCHAR(100) NOT NULL
);

CREATE TABLE Team (
   team_id INT PRIMARY KEY,
   team_name VARCHAR(100) NOT NULL,
   city VARCHAR(100) NOT NULL,
   wins INT DEFAULT 0,
   losses INT DEFAULT 0,
   ties INT DEFAULT 0
);

CREATE TABLE Player (
   player_id INT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   team_id INT,
   position VARCHAR(50),
   age INT,
   height DECIMAL(5,2),
   weight DECIMAL(5,2),
   FOREIGN KEY (team_id) REFERENCES Team(team_id)
);

CREATE TABLE FantasyTeam (
   fantasy_team_id INT PRIMARY KEY,
   user_id INT,
```

```
    fantasy_team_name VARCHAR(100) NOT NULL,
    roster_size INT,
    FOREIGN KEY (user_id) REFERENCES UserInfo(user_id)
);

CREATE TABLE Games (
    game_id INT PRIMARY KEY,
    home_team INT,
    away_team INT,
    points_scored INT,
    passing_yards INT,
    rushing_yards INT,
    receiving_yards INT,
    touchdowns INT,
    receptions INT,
    fumbles_lost INT,
    expected_points INT,
    player_id INT,
    FOREIGN KEY (player_id) REFERENCES Player(player_id)
);
```

3. **Insert data to tables 1000 rows in 3 different tables (can create python script for filling in some data**

Proof that our tables have over 1000 rows in them

```
mysql> SELECT COUNT(*) FROM FantasyTeam;
+----------+
| COUNT(*) |
+----------+
|     1001 |
+----------+
```

```
mysql> SELECT COUNT(*) FROM Player;
+----------+
| COUNT(*) |
+----------+
|     1172 |
+----------+
```

```
mysql> SELECT COUNT(*) FROM UserInfo;
+----------+
| COUNT(*) |
+----------+
|     1001 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Games;
+----------+
| COUNT(*) |
+----------+
|     9369 |
+----------+
1 row in set (0.00 sec)
```

```
mysql> SELECT COUNT(*) FROM Team;
+----------+
| COUNT(*) |
+----------+
|       33 |
+----------+
1 row in set (0.00 sec)
```

4. **Create 4 advanced queries relevant to functionality of application**

5. **Screenshots of the top 15 rows for queries above**

QUERY1: Works

Best player each position:

```
SELECT
   p.name AS player_name,
   p.position,
   t.team_name,
   SUM(g.points_scored) AS total_points
FROM
   Player p
JOIN
   Games g ON p.player_id = g.player_id
JOIN
   Team t ON p.team_id = t.team_id
GROUP BY
   p.player_id, p.name, p.position, t.team_name
HAVING
   SUM(g.points_scored) > (
      SELECT AVG(total_points)
      FROM (
         SELECT SUM(g2.points_scored) AS total_points
         FROM Player p2
         JOIN Games g2 ON p2.player_id = g2.player_id
         GROUP BY p2.player_id
      ) AS player_totals
   )
ORDER BY
   total_points DESC;
```

```
    -> HAVING
    ->     SUM(g.points_scored) > (
    ->         SELECT AVG(total_points)
    ->         FROM (
    ->             SELECT SUM(g2.points_scored) AS total_points
    ->             FROM Player p2
    ->             JOIN Games g2 ON p2.player_id = g2.player_id
    ->             GROUP BY p2.player_id
    ->         ) AS player_totals
    ->     )
    -> ORDER BY
    ->     total_points DESC;,
+----------------------+----------+------------+--------------+
| player_name          | position | team_name  | total_points |
+----------------------+----------+------------+--------------+
| Calais Campbell      | DE       | Dolphins   |          329 |
| K'Waun Williams      | CB       | Titans     |          323 |
| Drew Dalman          | C        | Bears      |          322 |
| Matt Henningsen      | DE       | Titans     |          322 |
| Eno Benjamin         | RB       | Rams       |          321 |
| Shaquil Barrett      | OLB      | Broncos    |          320 |
| Adetokunbo Ogundeji  | OLB      | Bears      |          319 |
| Bernhard Raimann     | T        | Chargers   |          316 |
| Jameis Winston       | QB       | Ravens     |          315 |
| Colby Parkinson      | TE       | Panthers   |          313 |
| Jelani Woods         | TE       | Chargers   |          310 |
| Frankie Luvu         | ILB      | Packers    |          309 |
| Cooper Kupp          | WR       | Lions      |          308 |
| Adam Gotsis          | DE       | Commanders |          308 |
| Arik Armstead        | DT       | Browns     |          308 |
| Tyler Lockett        | WR       | Panthers   |          306 |
| J.D. McKissic        | RB       | Seahawks   |          305 |
| Alohi Gilman         | SS       | Cardinals  |          305 |
| Trey Smith           | G        | Bills      |          305 |
```

This query is needed for our application because we obviously need the top scorers by position/team. That is a crucial fantasy metric and informs a lot about who to select since there will be one “starting” player for each time that is key.

6. QUERY2:

Highest Scoring Player: Works

```
SELECT
  team_name,
  player_name,
  total_points
FROM
  (SELECT
    T.team_name,
    P.name AS player_name,
    SUM(G.points_scored) AS total_points
  FROM
    Player P
  JOIN
    Team T ON P.team_id = T.team_id
  JOIN
    Games G ON P.player_id = G.player_id
  GROUP BY
    T.team_name, P.name
  HAVING
    SUM(G.points_scored) > 250) AS Points
WHERE
  (team_name, total_points) IN (
    SELECT
      team_name, MAX(total_points)
    FROM
      (SELECT
        T.team_name,
        P.name AS player_name,
        SUM(G.points_scored) AS total_points
      FROM
        Player P
      JOIN
        Team T ON P.team_id = T.team_id
      JOIN
        Games G ON P.player_id = G.player_id
      GROUP BY
        T.team_name, P.name
      HAVING
        SUM(G.points_scored) > 250) AS TeamPlayerPoints
      GROUP BY
        team_name
  )
ORDER BY
  team_name
LIMIT 15;
```

```
+------------+-----------------+--------------+
| team_name  | player_name     | total_points |
+------------+-----------------+--------------+
| 49ers      | Michael Carter  |          513 |
| Bears      | Drew Dalman     |          322 |
| Bengals    | Derek Watt      |          293 |
| Bills      | Trey Smith      |          305 |
| Broncos    | Shaquil Barrett |          320 |
| Browns     | Arik Armstead   |          308 |
| Buccaneers | Akayleb Evans   |          286 |
| Cardinals  | Alohi Gilman    |          305 |
| Chargers   | Bernhard Raimann|          316 |
| Chiefs     | DeVante Parker  |          305 |
| Colts      | Gregory Rousseau|          291 |
| Commanders | Adam Gotsis     |          308 |
| Cowboys    | Connor Williams |          304 |
| Dolphins   | Calais Campbell |          329 |
| Eagles     | Joe Mixon       |          282 |
+------------+-----------------+--------------+
15 rows in set (0.05 sec)

mysql>
```

This query is essential for our FantasyResearchAssistant project as it identifies the highest scoring players across different teams who have surpassed a threshold, this allows a fantasy player during a draft to select. By filtering with more than 250 total points, you can choose a better player and help the fantasy player make a informed, data driven decision.

QUERY3:

Most consistent players

```
SELECT p.name AS player_name, p.position, t.team_name,
   STDDEV(g.points_scored) AS points_consistency
FROM Player p
JOIN Games g ON p.player_id = g.player_id
JOIN Team t ON p.team_id = t.team_id
GROUP BY p.player_id, p.position, t.team_name
HAVING COUNT(g.game_id) > 1  -- Ensure player has multiple games for consistency measure
ORDER BY points_consistency ASC
LIMIT 15;
```

```
mysql> SELECT p.name AS player_name, p.position, t.team_name,
    ->     STDDEV(g.points_scored) AS points_consistency
    -> FROM Player p
    -> JOIN Games g ON p.player_id = g.player_id
    -> JOIN Team t ON p.team_id = t.team_id
    -> GROUP BY p.player_id, p.position, t.team_name
    -> HAVING COUNT(g.game_id) > 1  -- Ensure player has multiple games for consistency measure
    -> ORDER BY points_consistency ASC
    -> LIMIT 15;
+---------------------+----------+------------+--------------------+
| player_name         | position | team_name  | points_consistency |
+---------------------+----------+------------+--------------------+
| Adetokunbo Ogundeji | OLB      | Bears      |  4.044672421840859 |
| Leki Fotu           | DT       | Rams       |   4.19635258289863 |
| Adrian Amos         | FS       | Jaguars    |  4.539754949333719 |
| Josh Allen          | OLB      | Commanders |  4.621620386834037 |
| Jamal Agnew         | WR       | Commanders |  5.049752469181039 |
| George Fant         | T        | 49ers      |  5.158427570490837 |
| Carlton Davis       | CB       | Broncos    |   5.18260311040697 |
| Joel Bitonio        | G        | Jets       |  5.266343608235225 |
| Antoine Winfield    | FS       | Broncos    | 5.4184292004233106 |
| Chris Rumph         | OLB      | Cardinals  |  5.454356057317858 |
| Tedarrell Slaton    | DT       | Jaguars    |  5.454356057317858 |
| Shi Smith           | WR       | Packers    | 5.5396299515400855 |
| Amani Hooker        | FS       | Saints     |   5.62916512459885 |
| Micah Hyde          | FS       | Colts      |  5.717298313014636 |
| Avonte Maddox       | CB       | Raiders    |  5.722761571129799 |
+---------------------+----------+------------+--------------------+
15 rows in set (0.03 sec)
```

This query helps us determine the player's consistency so the users can be well informed when drafting their fantasy players. This query uses a player's games and points scored to calculate the standard deviation of the player's points scored. This helps us determine the most consistent players based on highest point consistency.

QUERY4: Each player's total fantasy points based on Fantasy Criteria
SELECT p.player_id, p.name, t.team_name AS team,
SUM((g.rushing_yards / 10) + (g.receiving_yards / 10) + (g.passing_yards / 25) + (g.touchdowns * 6) + (g.receptions * 1) - (g.fumbles_lost * 2))
AS total_fantasy_points
FROM Player p JOIN Games g ON p.player_id = g.player_id JOIN Team t ON p.team_id = t.team_id GROUP BY p.player_id, p.name, t.team_name
ORDER BY total_fantasy_points DESC;

```
mysql> SELECT
    ->     p.player_id,
    ->     p.name,
    ->     t.team_name AS team,
    ->     SUM(
    ->         (g.rushing_yards / 10) +
    ->         (g.receiving_yards / 10) +
    ->         (g.passing_yards / 25) +
    ->         (g.touchdowns * 6) +
    ->         (g.receptions * 1) -
    ->         (g.fumbles_lost * 2)
    ->     ) AS total_fantasy_points
    -> FROM
    ->     Player p
    -> JOIN
    ->     Games g ON p.player_id = g.player_id
    -> JOIN
    ->     Team t ON p.team_id = t.team_id
    -> GROUP BY
    ->     p.player_id, p.name, t.team_name
    -> ORDER BY
    ->     total_fantasy_points DESC
    -> LIMIT 15;
+-----------+-----------------+----------+----------------------+
| player_id | name            | team     | total_fantasy_points |
+-----------+-----------------+----------+----------------------+
|     53520 | Anthony Schwartz | Jets     |             468.6200 |
|     54503 | Arnold Ebiketie  | Bears    |             465.5200 |
|     54494 | Cole Strange     | Chiefs   |             459.9000 |
|     52557 | Danny Pinter     | Chargers |             455.1200 |
|     41242 | Zack Martin      | Falcons  |             450.5800 |
|     44931 | Mack Hollins     | Texans   |             449.2600 |
|     52425 | CeeDee Lamb      | Falcons  |             446.1200 |
|     53646 | Khalil Herbert   | Vikings  |             441.8800 |
|     54571 | Cade Otton       | Broncos  |             441.6600 |
|     38544 | Michael Brockers | Patriots |             441.5600 |
|     42400 | Rob Havenstein   | Lions    |             440.5600 |
|     43413 | Deon Bush        | Bills    |             439.9800 |
|     54905 | Jaylen Warren    | Bengals  |             439.1200 |
|     41939 | Andrew Norwell   | Seahawks |             438.8800 |
|     42827 | Justin Coleman   | Panthers |             438.6400 |
+-----------+-----------------+----------+----------------------+
15 rows in set (0.07 sec)
```

This query is useful for fantasy football players because it calculates each player's total fantasy points across all games, based on standard fantasy football scoring rules. Allows us to identify top performers, spot trends in player performances, and evaluate trades. With an up-to-date

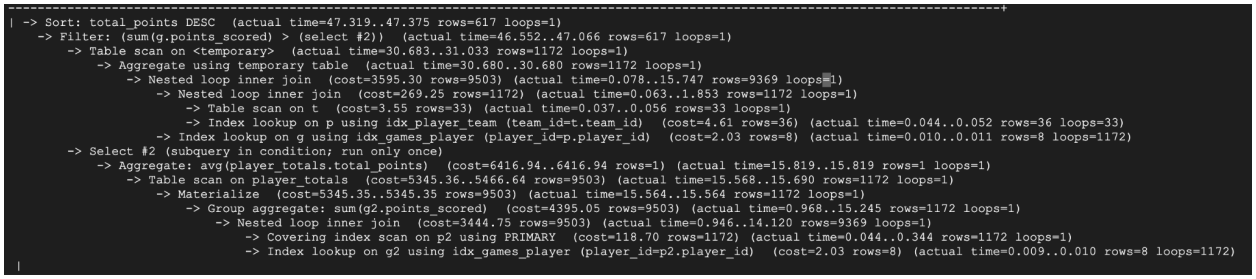
leaderboard, players can understand in what areas they are lacking in and plan ahead for the future.

Pt 2 indexing:

1. **Analyze/Explain command performance**
2. **Explore tradeoffs**
3. **Report final index design**

QUERY1:

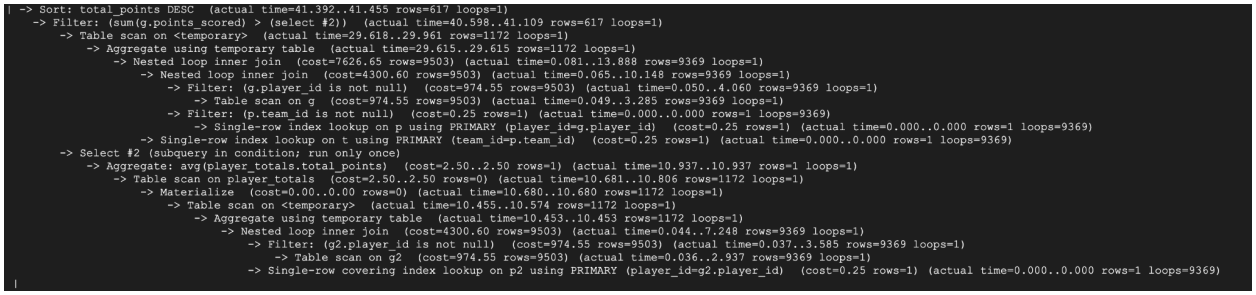
Running it bare bones no indexing output:



```
| -> Sort: total_points DESC  (actual time=47.319..47.375 rows=617 loops=1)
    -> Filter: (sum(g.points_scored) > (select #2))  (actual time=46.552..47.066 rows=617 loops=1)
        -> Table scan on <temporary>  (actual time=30.683..31.033 rows=1172 loops=1)
            -> Aggregate using temporary table  (actual time=30.680..30.680 rows=1172 loops=1)
                -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.078..15.747 rows=9369 loops=1)
                    -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.063..1.853 rows=1172 loops=1)
                        -> Table scan on t  (cost=3.55 rows=33) (actual time=0.037..0.056 rows=33 loops=1)
                        -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.044..0.052 rows=36 loops=33)
                    -> Index lookup on g using idx_games_player (player_id=p.player_id)  (cost=2.03 rows=8) (actual time=0.010..0.011 rows=8 loops=1172)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(player_totals.total_points)  (cost=6416.94..6416.94 rows=1) (actual time=15.819..15.819 rows=1 loops=1)
                -> Table scan on player_totals  (cost=5345.36..5466.64 rows=9503) (actual time=15.568..15.690 rows=1172 loops=1)
                    -> Materialize  (cost=5345.35..5345.35 rows=9503) (actual time=15.564..15.564 rows=1172 loops=1)
                        -> Group aggregate: sum(g2.points_scored)  (cost=4395.05 rows=9503) (actual time=0.968..15.245 rows=1172 loops=1)
                            -> Nested loop inner join  (cost=3444.75 rows=9503) (actual time=0.946..14.120 rows=9369 loops=1)
                                -> Covering index scan on p2 using PRIMARY  (cost=118.70 rows=1172) (actual time=0.044..0.344 rows=1172 loops=1)
                                -> Index lookup on g2 using idx_games_player (player_id=p2.player_id)  (cost=2.03 rows=8) (actual time=0.009..0.010 rows=8 loops=1172)
|
```

Some things I noticed here were that the queries within select #2 took most of the time. So I'm going to try strategies to try and reduce that.

Strategy 1: CREATE INDEX idx_points_scored ON Games(points_scored);



```
| -> Sort: total_points DESC  (actual time=41.392..41.455 rows=617 loops=1)
    -> Filter: (sum(g.points_scored) > (select #2))  (actual time=40.598..41.109 rows=617 loops=1)
        -> Table scan on <temporary>  (actual time=29.618..29.961 rows=1172 loops=1)
            -> Aggregate using temporary table  (actual time=29.615..29.615 rows=1172 loops=1)
                -> Nested loop inner join  (cost=7626.65 rows=9503) (actual time=0.081..13.888 rows=9369 loops=1)
                    -> Nested loop inner join  (cost=4300.60 rows=9503) (actual time=0.065..10.148 rows=9369 loops=1)
                        -> Filter: (g.player_id is not null)  (cost=974.55 rows=9503) (actual time=0.050..4.060 rows=9369 loops=1)
                            -> Table scan on g  (cost=974.55 rows=9503) (actual time=0.049..3.285 rows=9369 loops=1)
                        -> Filter: (p.team_id is not null)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=9369)
                            -> Single-row index lookup on p using PRIMARY (player_id=g.player_id)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=9369)
                    -> Single-row index lookup on t using PRIMARY (team_id=p.team_id)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=9369)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(player_totals.total_points)  (cost=2.50..2.50 rows=1) (actual time=10.937..10.937 rows=1 loops=1)
                -> Table scan on player_totals  (cost=2.50..2.50 rows=0) (actual time=10.681..10.806 rows=1172 loops=1)
                    -> Materialize  (cost=0.00..0.00 rows=0) (actual time=10.680..10.680 rows=1172 loops=1)
                        -> Table scan on <temporary>  (actual time=10.455..10.574 rows=1172 loops=1)
                            -> Aggregate using temporary table  (actual time=10.453..10.453 rows=1172 loops=1)
                                -> Nested loop inner join  (cost=4300.60 rows=9503) (actual time=0.044..7.248 rows=9369 loops=1)
                                    -> Filter: (g2.player_id is not null)  (cost=974.55 rows=9503) (actual time=0.037..3.585 rows=9369 loops=1)
                                        -> Table scan on g2  (cost=974.55 rows=9503) (actual time=0.036..2.937 rows=9369 loops=1)
                                    -> Single-row covering index lookup on p2 using PRIMARY (player_id=g2.player_id)  (cost=0.25 rows=1) (actual time=0.000..0.000 rows=1 loops=9369)
|
```

We can see this sped up the query a lot ~50%. Cost dropped as well for where the index helped out.

Strategy 2: CREATE INDEX idx_team_name ON Team(team_name);

```
| -> Sort: total_points DESC  (actual time=49.125..49.292 rows=617 loops=1)
    -> Filter: (sum(g.points_scored) > (select #2))  (actual time=47.486..48.212 rows=617 loops=1)
        -> Table scan on <temporary>  (actual time=30.614..31.091 rows=1172 loops=1)
            -> Aggregate using temporary table  (actual time=30.611..30.611 rows=1172 loops=1)
                -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.138..15.918 rows=9369 loops=1)
                    -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.115..1.790 rows=1172 loops=1)
                        -> Covering index scan on t using idx_team_name  (cost=3.55 rows=33) (actual time=0.037..0.052 rows=33 loops=1)
                        -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.043..0.050 rows=36 loops=33)
                    -> Index lookup on g using idx_games_player (player_id=p.player_id)  (cost=2.03 rows=8) (actual time=0.010..0.011 rows=8 loops=1172)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(player_totals.total_points)  (cost=6416.94..6416.94 rows=1) (actual time=15.524..15.525 rows=1 loops=1)
                -> Table scan on player_totals  (cost=5345.36..5466.64 rows=9503) (actual time=15.274..15.395 rows=1172 loops=1)
                    -> Materialize  (cost=5345.35..5345.35 rows=9503) (actual time=15.270..15.270 rows=1172 loops=1)
                        -> Group aggregate: sum(g2.points_scored)  (cost=4395.05 rows=9503) (actual time=0.102..14.058 rows=1172 loops=1)
                            -> Nested loop inner join  (cost=3444.75 rows=9503) (actual time=0.077..12.907 rows=9369 loops=1)
                                -> Covering index scan on p2 using PRIMARY  (cost=118.70 rows=1172) (actual time=0.052..0.350 rows=1172 loops=1)
                                -> Index lookup on g2 using idx_games_player (player_id=p2.player_id)  (cost=2.03 rows=8) (actual time=0.009..0.010 rows=8 loops=1172)
 |
```

This query did not actually lead to much speed up at all, it still took the same time to run as the original. We do see that the cost was diminished however.

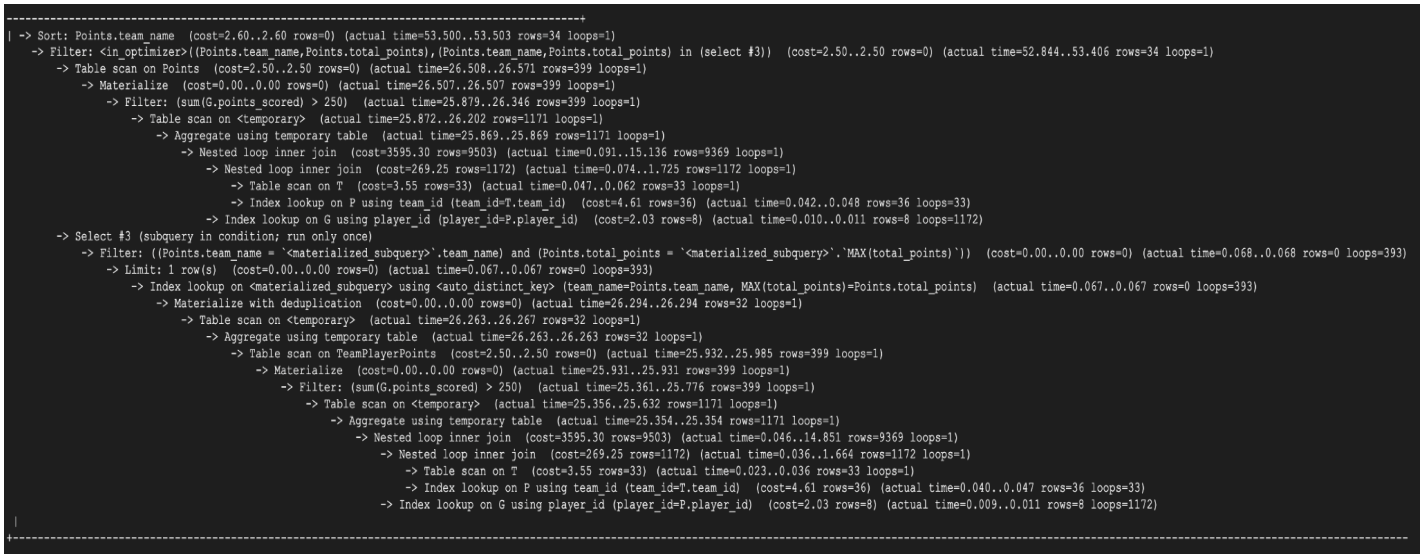
Strategy 3: CREATE INDEX idx_player_points ON Games(player_id, points_scored);

```
| -> Sort: total_points DESC  (actual time=54.482..54.583 rows=617 loops=1)
    -> Filter: (sum(g.points_scored) > (select #2))  (actual time=53.197..53.918 rows=617 loops=1)
        -> Table scan on <temporary>  (actual time=40.248..40.682 rows=1172 loops=1)
            -> Aggregate using temporary table  (actual time=40.244..40.244 rows=1172 loops=1)
                -> Nested loop inner join  (cost=2324.89 rows=9503) (actual time=0.204..14.818 rows=9369 loops=1)
                    -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.180..3.039 rows=1172 loops=1)
                        -> Covering index scan on t using idx_team_name  (cost=3.55 rows=33) (actual time=0.073..0.112 rows=33 loops=1)
                        -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.073..0.085 rows=36 loops=33)
                    -> Covering index lookup on g using idx_player_points (player_id=p.player_id)  (cost=0.94 rows=8) (actual time=0.007..0.009 rows=8 loops=1172)
        -> Select #2 (subquery in condition; run only once)
            -> Aggregate: avg(player_totals.total_points)  (cost=5146.53..5146.53 rows=1) (actual time=12.901..12.901 rows=1 loops=1)
                -> Table scan on player_totals  (cost=4074.95..4196.23 rows=9503) (actual time=12.469..12.646 rows=1172 loops=1)
                    -> Materialize  (cost=4074.94..4074.94 rows=9503) (actual time=12.465..12.465 rows=1172 loops=1)
                        -> Group aggregate: sum(g2.points_scored)  (cost=3124.64 rows=9503) (actual time=0.070..11.918 rows=1172 loops=1)
                            -> Nested loop inner join  (cost=2174.34 rows=9503) (actual time=0.056..9.869 rows=9369 loops=1)
                                -> Covering index scan on p2 using PRIMARY  (cost=118.70 rows=1172) (actual time=0.045..0.565 rows=1172 loops=1)
                                -> Covering index lookup on g2 using idx_player_points (player_id=p2.player_id)  (cost=0.94 rows=8) (actual time=0.005..0.007 rows=8 loops=1172)
 |
```

I expected this strategy to work the best but it did the same as the second strategy in terms of time and cost performance.

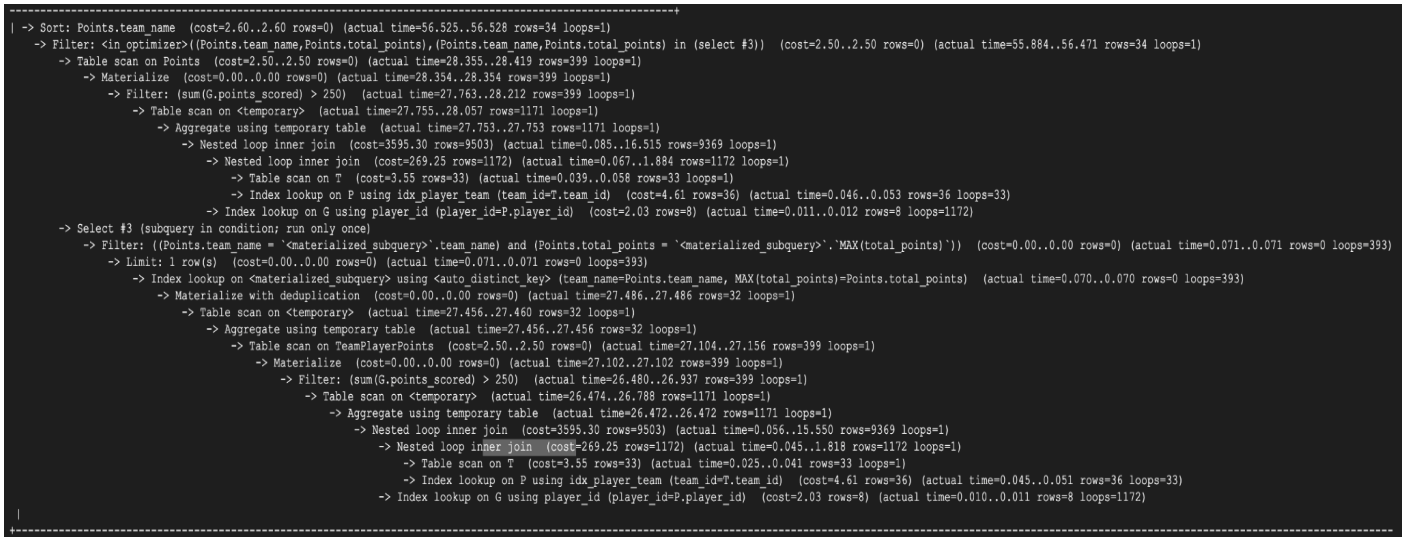
Overall the best strategy for indexing for the first query is indexing based off of points scored in games.

QUERY2:
Running query 2 normally with explain analyze



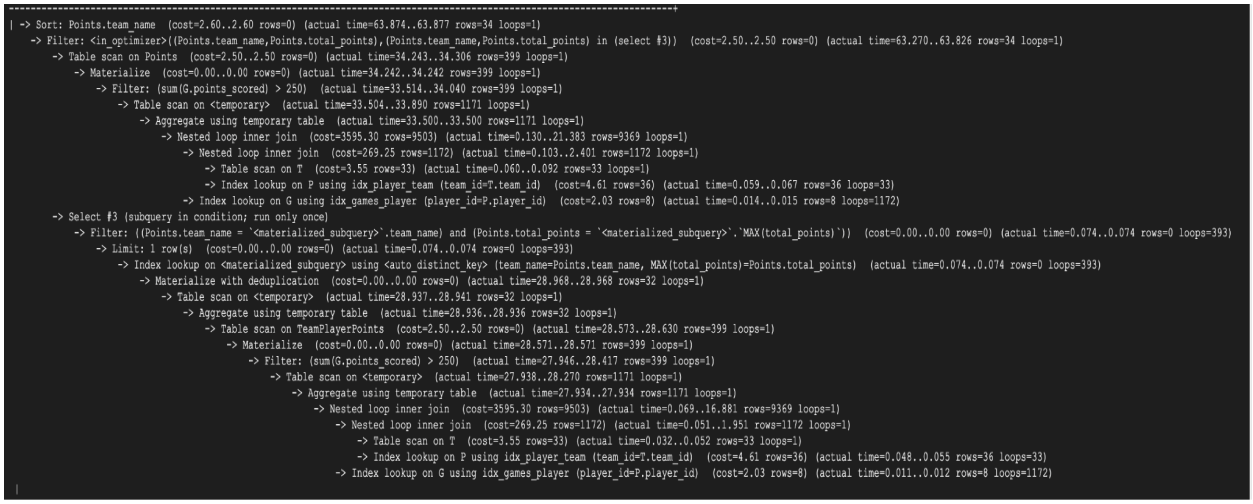
```
-> Sort: Points.team_name  (cost=2.60..2.60 rows=0) (actual time=53.500..53.503 rows=34 loops=1)
    -> Filter: <in_optimizer>((Points.team_name,Points.total_points),(Points.team_name,Points.total_points) in (select #3))  (cost=2.50..2.50 rows=0) (actual time=52.844..53.406 rows=34 loops=1)
        -> Table scan on Points  (cost=2.50..2.50 rows=0) (actual time=26.508..26.571 rows=399 loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=26.507..26.507 rows=399 loops=1)
                -> Filter: (sum(G.points_scored) > 250)  (actual time=25.879..26.346 rows=399 loops=1)
                    -> Table scan on <temporary>  (actual time=25.872..26.202 rows=1171 loops=1)
                        -> Aggregate using temporary table  (actual time=25.869..25.869 rows=1171 loops=1)
                            -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.091..15.136 rows=9369 loops=1)
                                -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.074..1.725 rows=1172 loops=1)
                                    -> Table scan on T  (cost=3.55 rows=33) (actual time=0.047..0.062 rows=33 loops=1)
                                    -> Index lookup on P using team_id (team_id=T.team_id)  (cost=4.61 rows=36) (actual time=0.042..0.048 rows=36 loops=33)
                                -> Index lookup on G using player_id (player_id=P.player_id)  (cost=2.03 rows=8) (actual time=0.010..0.011 rows=8 loops=1172)
        -> Select #3 (subquery in condition; run only once)
            -> Filter: ((Points.team_name = `<materialized_subquery>`.team_name) and (Points.total_points = `<materialized_subquery>`.`MAX(total_points)`))  (cost=0.00..0.00 rows=0) (actual time=0.068..0.068 rows=0 loops=393)
                -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.067..0.067 rows=0 loops=393)
                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (team_name=Points.team_name, MAX(total_points)=Points.total_points)  (actual time=0.067..0.067 rows=0 loops=393)
                        -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=26.294..26.294 rows=32 loops=1)
                            -> Table scan on <temporary>  (actual time=26.263..26.267 rows=32 loops=1)
                                -> Aggregate using temporary table  (actual time=26.263..26.263 rows=32 loops=1)
                                    -> Table scan on TeamPlayerPoints  (cost=2.50..2.50 rows=0) (actual time=25.932..25.985 rows=399 loops=1)
                                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=25.931..25.931 rows=399 loops=1)
                                            -> Filter: (sum(G.points_scored) > 250)  (actual time=25.361..25.776 rows=399 loops=1)
                                                -> Table scan on <temporary>  (actual time=25.356..25.632 rows=1171 loops=1)
                                                    -> Aggregate using temporary table  (actual time=25.354..25.354 rows=1171 loops=1)
                                                        -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.046..14.851 rows=9369 loops=1)
                                                            -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.036..1.664 rows=1172 loops=1)
                                                                -> Table scan on T  (cost=3.55 rows=33) (actual time=0.023..0.036 rows=33 loops=1)
                                                                -> Index lookup on P using team_id (team_id=T.team_id)  (cost=4.61 rows=36) (actual time=0.040..0.047 rows=36 loops=33)
                                                            -> Index lookup on G using player_id (player_id=P.player_id)  (cost=2.03 rows=8) (actual time=0.009..0.011 rows=8 loops=1172)
```

We tried first index CREATE INDEX idx_player_team ON Player(team_id);
This index was not beneficial and actually made the query run 3 ms slower (actual time=56.525 > 53.500)



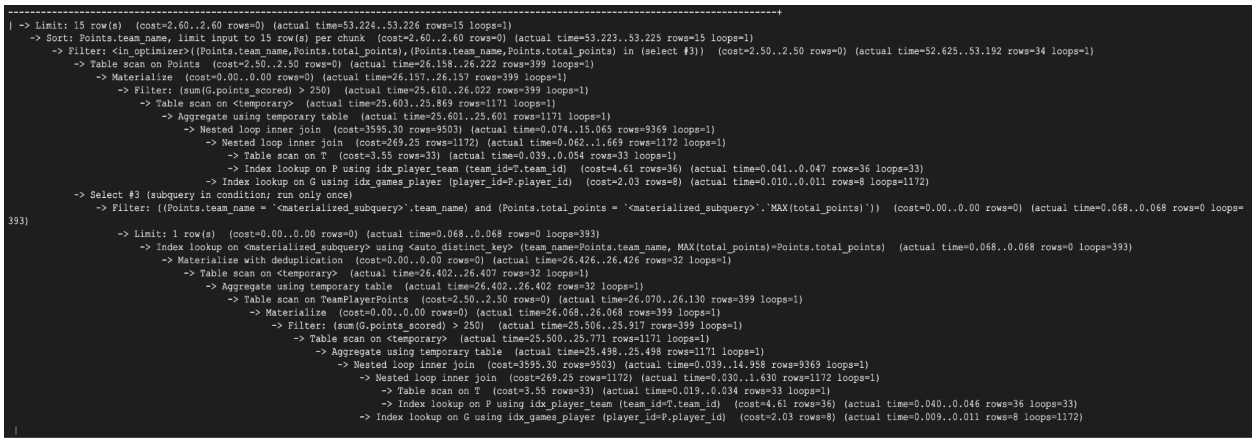
```
-> Sort: Points.team_name  (cost=2.60..2.60 rows=0) (actual time=56.525..56.528 rows=34 loops=1)
    -> Filter: <in_optimizer>((Points.team_name,Points.total_points),(Points.team_name,Points.total_points) in (select #3))  (cost=2.50..2.50 rows=0) (actual time=55.884..56.471 rows=34 loops=1)
        -> Table scan on Points  (cost=2.50..2.50 rows=0) (actual time=28.355..28.419 rows=399 loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=28.354..28.354 rows=399 loops=1)
                -> Filter: (sum(G.points_scored) > 250)  (actual time=27.763..28.212 rows=399 loops=1)
                    -> Table scan on <temporary>  (actual time=27.755..28.057 rows=1171 loops=1)
                        -> Aggregate using temporary table  (actual time=27.753..27.753 rows=1171 loops=1)
                            -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.085..16.515 rows=9369 loops=1)
                                -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.067..1.884 rows=1172 loops=1)
                                    -> Table scan on T  (cost=3.55 rows=33) (actual time=0.039..0.058 rows=33 loops=1)
                                    -> Index lookup on P using idx_player_team (team_id=T.team_id)  (cost=4.61 rows=36) (actual time=0.046..0.053 rows=36 loops=33)
                                -> Index lookup on G using player_id (player_id=P.player_id)  (cost=2.03 rows=8) (actual time=0.011..0.012 rows=8 loops=1172)
        -> Select #3 (subquery in condition; run only once)
            -> Filter: ((Points.team_name = `<materialized_subquery>`.team_name) and (Points.total_points = `<materialized_subquery>`.`MAX(total_points)`))  (cost=0.00..0.00 rows=0) (actual time=0.071..0.071 rows=0 loops=393)
                -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.071..0.071 rows=0 loops=393)
                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (team_name=Points.team_name, MAX(total_points)=Points.total_points)  (actual time=0.070..0.070 rows=0 loops=393)
                        -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=27.486..27.486 rows=32 loops=1)
                            -> Table scan on <temporary>  (actual time=27.456..27.460 rows=32 loops=1)
                                -> Aggregate using temporary table  (actual time=27.456..27.456 rows=32 loops=1)
                                    -> Table scan on TeamPlayerPoints  (cost=2.50..2.50 rows=0) (actual time=27.104..27.156 rows=399 loops=1)
                                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=27.102..27.102 rows=399 loops=1)
                                            -> Filter: (sum(G.points_scored) > 250)  (actual time=26.480..26.937 rows=399 loops=1)
                                                -> Table scan on <temporary>  (actual time=26.474..26.788 rows=1171 loops=1)
                                                    -> Aggregate using temporary table  (actual time=26.472..26.472 rows=1171 loops=1)
                                                        -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.056..15.550 rows=9369 loops=1)
                                                            -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.045..1.818 rows=1172 loops=1)
                                                                -> Table scan on T  (cost=3.55 rows=33) (actual time=0.025..0.041 rows=33 loops=1)
                                                                -> Index lookup on P using idx_player_team (team_id=T.team_id)  (cost=4.61 rows=36) (actual time=0.045..0.051 rows=36 loops=33)
                                                            -> Index lookup on G using player_id (player_id=P.player_id)  (cost=2.03 rows=8) (actual time=0.010..0.011 rows=8 loops=1172)
```

The next index we tried was CREATE INDEX idx_games_player ON Games(player_id);
This index was not beneficial since the time increased to 63.874 from 53.50 so we will not use it



```
-> Sort: Points.team_name  (cost=2.60..2.60 rows=0) (actual time=63.874..63.877 rows=34 loops=1)
    -> Filter: <in_optimizer>((Points.team_name,Points.total_points),(Points.team_name,Points.total_points) in (select #3))  (cost=2.50..2.50 rows=0) (actual time=63.270..63.826 rows=34 loops=1)
        -> Table scan on Points  (cost=2.50..2.50 rows=0) (actual time=34.243..34.306 rows=399 loops=1)
            -> Materialize  (cost=0.00..0.00 rows=0) (actual time=34.242..34.242 rows=399 loops=1)
                -> Filter: (sum(G.points_scored) > 250)  (actual time=33.514..34.040 rows=399 loops=1)
                    -> Table scan on <temporary>  (actual time=33.504..33.890 rows=1171 loops=1)
                        -> Aggregate using temporary table  (actual time=33.500..33.500 rows=1171 loops=1)
                            -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.130..21.383 rows=9369 loops=1)
                                -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.103..2.401 rows=1172 loops=1)
                                    -> Table scan on T  (cost=3.55 rows=33) (actual time=0.060..0.092 rows=33 loops=1)
                                    -> Index lookup on P using idx_player_team (team_id=T.team_id)  (cost=4.61 rows=36) (actual time=0.059..0.067 rows=36 loops=33)
                                -> Index lookup on G using idx_games_player (player_id=P.player_id)  (cost=2.03 rows=8) (actual time=0.014..0.015 rows=8 loops=1172)
        -> Select #3 (subquery in condition; run only once)
            -> Filter: ((Points.team_name = `<materialized_subquery>`.team_name) and (Points.total_points = `<materialized_subquery>`.`MAX(total_points)`))  (cost=0.00..0.00 rows=0) (actual time=0.074..0.074 rows=0 loops=393)
                -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.074..0.074 rows=0 loops=393)
                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (team_name=Points.team_name, MAX(total_points)=Points.total_points)  (actual time=0.074..0.074 rows=0 loops=393)
                        -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=28.968..28.968 rows=32 loops=1)
                            -> Table scan on <temporary>  (actual time=28.937..28.941 rows=32 loops=1)
                                -> Aggregate using temporary table  (actual time=28.936..28.936 rows=32 loops=1)
                                    -> Table scan on TeamPlayerPoints  (cost=2.50..2.50 rows=0) (actual time=28.573..28.630 rows=399 loops=1)
                                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=28.571..28.571 rows=399 loops=1)
                                            -> Filter: (sum(G.points_scored) > 250)  (actual time=27.946..28.417 rows=399 loops=1)
                                                -> Table scan on <temporary>  (actual time=27.938..28.270 rows=1171 loops=1)
                                                    -> Aggregate using temporary table  (actual time=27.934..27.934 rows=1171 loops=1)
                                                        -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.069..16.881 rows=9369 loops=1)
                                                            -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.051..1.951 rows=1172 loops=1)
                                                                -> Table scan on T  (cost=3.55 rows=33) (actual time=0.032..0.052 rows=33 loops=1)
                                                                -> Index lookup on P using idx_player_team (team_id=T.team_id)  (cost=4.61 rows=36) (actual time=0.048..0.055 rows=36 loops=33)
                                                            -> Index lookup on G using idx_games_player (player_id=P.player_id)  (cost=2.03 rows=8) (actual time=0.011..0.012 rows=8 loops=1172)
```

The last index we tried was
CREATE INDEX idx_position ON Player(position);
CREATE INDEX idx_age ON Player(age);
This is because we are querying a lot around this information in the subqueries so we tried seeing if it would increase performance.



```
-> Limit: 15 row(s)  (cost=2.60..2.60 rows=0) (actual time=53.224..53.226 rows=15 loops=1)
    -> Sort: Points.team_name, limit input to 15 row(s) per chunk  (cost=2.60..2.60 rows=0) (actual time=53.223..53.225 rows=15 loops=1)
        -> Filter: <in_optimizer>((Points.team_name,Points.total_points),(Points.team_name,Points.total_points) in (select #3))  (cost=2.50..2.50 rows=0) (actual time=52.625..53.192 rows=34 loops=1)
            -> Table scan on Points  (cost=2.50..2.50 rows=0) (actual time=26.158..26.222 rows=399 loops=1)
                -> Materialize  (cost=0.00..0.00 rows=0) (actual time=26.157..26.157 rows=399 loops=1)
                    -> Filter: (sum(G.points_scored) > 250)  (actual time=25.610..26.022 rows=399 loops=1)
                        -> Table scan on <temporary>  (actual time=25.603..25.869 rows=1171 loops=1)
                            -> Aggregate using temporary table  (actual time=25.601..25.601 rows=1171 loops=1)
                                -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.074..15.065 rows=9369 loops=1)
                                    -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.062..1.669 rows=1172 loops=1)
                                        -> Table scan on T  (cost=3.55 rows=33) (actual time=0.039..0.054 rows=33 loops=1)
                                        -> Index lookup on P using idx_player_team (team_id=T.team_id)  (cost=4.61 rows=36) (actual time=0.041..0.047 rows=36 loops=33)
                                    -> Index lookup on G using idx_games_player (player_id=P.player_id)  (cost=2.03 rows=8) (actual time=0.010..0.011 rows=8 loops=1172)
        -> Select #3 (subquery in condition; run only once)
            -> Filter: ((Points.team_name = `<materialized_subquery>`.team_name) and (Points.total_points = `<materialized_subquery>`.`MAX(total_points)`))  (cost=0.00..0.00 rows=0) (actual time=0.068..0.068 rows=0 loops=
393)
                -> Limit: 1 row(s)  (cost=0.00..0.00 rows=0) (actual time=0.068..0.068 rows=0 loops=393)
                    -> Index lookup on <materialized_subquery> using <auto_distinct_key> (team_name=Points.team_name, MAX(total_points)=Points.total_points)  (actual time=0.068..0.068 rows=0 loops=393)
                        -> Materialize with deduplication  (cost=0.00..0.00 rows=0) (actual time=26.426..26.426 rows=32 loops=1)
                            -> Table scan on <temporary>  (actual time=26.402..26.407 rows=32 loops=1)
                                -> Aggregate using temporary table  (actual time=26.402..26.402 rows=32 loops=1)
                                    -> Table scan on TeamPlayerPoints  (cost=2.50..2.50 rows=0) (actual time=26.070..26.130 rows=399 loops=1)
                                        -> Materialize  (cost=0.00..0.00 rows=0) (actual time=26.068..26.068 rows=399 loops=1)
                                            -> Filter: (sum(G.points_scored) > 250)  (actual time=25.506..25.917 rows=399 loops=1)
                                                -> Table scan on <temporary>  (actual time=25.500..25.771 rows=1171 loops=1)
                                                    -> Aggregate using temporary table  (actual time=25.498..25.498 rows=1171 loops=1)
                                                        -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.039..14.958 rows=9369 loops=1)
                                                            -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.030..1.630 rows=1172 loops=1)
                                                                -> Table scan on T  (cost=3.55 rows=33) (actual time=0.019..0.034 rows=33 loops=1)
                                                                -> Index lookup on P using idx_player_team (team_id=T.team_id)  (cost=4.61 rows=36) (actual time=0.040..0.046 rows=36 loops=33)
                                                            -> Index lookup on G using idx_games_player (player_id=P.player_id)  (cost=2.03 rows=8) (actual time=0.009..0.011 rows=8 loops=1172)
```

This did not increase the performance of our code since the time increased, for this given query we will choose not to index.

QUERY3:

```
mysql> explain analyze SELECT p.name AS player_name, p.position, t.team_name,
    ->     STDDEV(g.points_scored) AS points_consistency
    -> FROM Player p
    -> JOIN Games g ON p.player_id = g.player_id
    -> JOIN Team t ON p.team_id = t.team_id
    -> GROUP BY p.player_id, p.position, t.team_name
    -> HAVING COUNT(g.game_id) > 1  -- Ensure player has multiple games for consistency measure
    -> ORDER BY points_consistency ASC
    -> LIMIT 15;
+------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
--------------------+
| EXPLAIN




                                                                                                   |
+------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
--------------------+
| -> Limit: 15 row(s)  (actual time=1258.757..1258.761 rows=15 loops=1)
    -> Sort: points_consistency  (actual time=1258.756..1258.759 rows=15 loops=1)
        -> Filter: (count(g.game_id) > 1)  (actual time=1257.512..1258.144 rows=1171 loops=1)
            -> Table scan on <temporary>  (actual time=1257.493..1257.939 rows=1172 loops=1)
                -> Aggregate using temporary table  (actual time=1257.490..1257.490 rows=1172 loops=1)
                    -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.091..780.333 rows=9369 loops=1)
                        -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.074..12.498 rows=1172 loops=1)
                            -> Table scan on t  (cost=3.55 rows=33) (actual time=0.046..0.104 rows=33 loops=1)
                            -> Index lookup on p using team_id (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.139..0.372 rows=36 loops=33)
                        -> Index lookup on g using player_id (player_id=p.player_id)  (cost=2.03 rows=8) (actual time=0.646..0.654 rows=8 loops=1172)
 |
+------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
```

First, we tried to index on player_id: CREATE INDEX idx_player_id ON Player(player_id). This index did not change the cost of the query at all. So, we will choose not to use this index.

```
mysql> explain analyze SELECT p.name AS player_name, p.position, t.team_name,
    ->     STDDEV(g.points_scored) AS points_consistency
    -> FROM Player p
    -> JOIN Games g ON p.player_id = g.player_id
    -> JOIN Team t ON p.team_id = t.team_id
    -> GROUP BY p.player_id, p.position, t.team_name
    -> HAVING COUNT(g.game_id) > 1  -- Ensure player has multiple games for consistency measure
    -> ORDER BY points_consistency ASC
    -> LIMIT 15;
+-------------------------------------------------------------------------------------------------------------------------------------
-------------+
| EXPLAIN
                                                                                                                            |
+-------------------------------------------------------------------------------------------------------------------------------------
-------------+
| -> Limit: 15 row(s)  (actual time=26.678..26.681 rows=15 loops=1)
    -> Sort: points_consistency  (actual time=26.677..26.679 rows=15 loops=1)
        -> Filter: (count(g.game_id) > 1)  (actual time=25.609..26.052 rows=1171 loops=1)
            -> Table scan on <temporary>  (actual time=25.603..25.936 rows=1172 loops=1)
                -> Aggregate using temporary table  (actual time=25.600..25.600 rows=1172 loops=1)
                    -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.054..15.126 rows=9369 loops=1)
                        -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.043..1.799 rows=1172 loops=1)
                            -> Table scan on t  (cost=3.55 rows=33) (actual time=0.021..0.036 rows=33 loops=1)
                            -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.045..0.051 rows=36 loops=33)
                        -> Index lookup on g using idx_games_player (player_id=p.player_id)  (cost=2.03 rows=8) (actual time=0.010..0.011 rows=8 loops=1172)
 |
+-------------------------------------------------------------------------------------------------------------------------------------
```

Second, we tried to index team_id: CREATE INDEX idx_team_id ON Team(team_id). This index also did not change the cost of the query at all. So, we will choose not to use this index.

```
mysql> explain analyze SELECT p.name AS player_name, p.position, t.team_name,
    ->     STDDEV(g.points_scored) AS points_consistency
    -> FROM Player p
    -> JOIN Games g ON p.player_id = g.player_id
    -> JOIN Team t ON p.team_id = t.team_id
    -> GROUP BY p.player_id, p.position, t.team_name
    -> HAVING COUNT(g.game_id) > 1  -- Ensure player has multiple games for consistency measure
    -> ORDER BY points_consistency ASC
    -> LIMIT 15;
+-------------------------------------------------------------------------------------------------------------------------------------
-------------+
| EXPLAIN
                                                                                                                            |
+-------------------------------------------------------------------------------------------------------------------------------------
-------------+
| -> Limit: 15 row(s)  (actual time=28.984..28.986 rows=15 loops=1)
    -> Sort: points_consistency  (actual time=28.983..28.984 rows=15 loops=1)
        -> Filter: (count(g.game_id) > 1)  (actual time=27.700..28.241 rows=1171 loops=1)
            -> Table scan on <temporary>  (actual time=27.692..28.128 rows=1172 loops=1)
                -> Aggregate using temporary table  (actual time=27.688..27.688 rows=1172 loops=1)
                    -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.086..16.766 rows=9369 loops=1)
                        -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.070..1.996 rows=1172 loops=1)
                            -> Table scan on t  (cost=3.55 rows=33) (actual time=0.026..0.050 rows=33 loops=1)
                            -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.049..0.056 rows=36 loops=33)
                        -> Index lookup on g using idx_games_player (player_id=p.player_id)  (cost=2.03 rows=8) (actual time=0.011..0.012 rows=8 loops=1172)
 |
+-------------------------------------------------------------------------------------------------------------------------------------
```

Third, we tried to index the player position: CREATE INDEX idx_position ON Player(position). This index also did not change the cost of the query at all. So, we will choose not to use this index.

```
mysql> explain analyze SELECT p.name AS player_name, p.position, t.team_name,
    ->     STDDEV(g.points_scored) AS points_consistency
    -> FROM Player p
    -> JOIN Games g ON p.player_id = g.player_id
    -> JOIN Team t ON p.team_id = t.team_id
    -> GROUP BY p.player_id, p.position, t.team_name
    -> HAVING COUNT(g.game_id) > 1  -- Ensure player has multiple games for consistency measure
    -> ORDER BY points_consistency ASC
    -> LIMIT 15;
+--------------------------------------------------------------------------------------------------
------------------------------------------+
| EXPLAIN
                                                                                                  |
+--------------------------------------------------------------------------------------------------
------------------------------------------+
| -> Limit: 15 row(s)  (actual time=27.710..27.712 rows=15 loops=1)
    -> Sort: points_consistency  (actual time=27.709..27.710 rows=15 loops=1)
        -> Filter: (count(g.game_id) > 1)  (actual time=26.924..27.324 rows=1171 loops=1)
            -> Table scan on <temporary>  (actual time=26.919..27.205 rows=1172 loops=1)
                -> Aggregate using temporary table  (actual time=26.915..26.915 rows=1172 loops=1)
                    -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.822..16.366 rows=9369 loops=1)
                        -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.799..2.496 rows=1172 loops=1)
                            -> Covering index scan on t using idx_team_name  (cost=3.55 rows=33) (actual time=0.044..0.060 rows=33 loops=1)
                            -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.044..0.051 rows=36 loops=33)
                        -> Index lookup on g using idx_games_player (player_id=p.player_id)  (cost=2.03 rows=8) (actual time=0.010..0.011 rows=8 loops=1172)
 |
+--------------------------------------------------------------------------------------------------
```

Finally, we tried to index the team_name: CREATE INDEX idx_teamname ON Team(team_name). This index lowered our cost of the next loop inner join to 2324.89 from 3595.30. So, this is a good index to use.

```
mysql> explain analyze SELECT p.name AS player_name, p.position, t.team_name,
    ->     STDDEV(g.points_scored) AS points_consistency
    -> FROM Player p
    -> JOIN Games g ON p.player_id = g.player_id
    -> JOIN Team t ON p.team_id = t.team_id
    -> GROUP BY p.player_id, p.position, t.team_name
    -> HAVING COUNT(g.game_id) > 1  -- Ensure player has multiple games for consistency measure
    -> ORDER BY points_consistency ASC
    -> LIMIT 15;
+--------------------------------------------------------------------------------------------------
----------------------------------------------------+
| EXPLAIN
                                                                                                  |
+--------------------------------------------------------------------------------------------------
----------------------------------------------------+
| -> Limit: 15 row(s)  (actual time=19.661..19.664 rows=15 loops=1)
    -> Sort: points_consistency  (actual time=19.660..19.662 rows=15 loops=1)
        -> Filter: (count(g.game_id) > 1)  (actual time=18.867..19.266 rows=1171 loops=1)
            -> Table scan on <temporary>  (actual time=18.862..19.153 rows=1172 loops=1)
                -> Aggregate using temporary table  (actual time=18.858..18.858 rows=1172 loops=1)
                    -> Nested loop inner join  (cost=2324.89 rows=9503) (actual time=0.808..7.929 rows=9369 loops=1)
                        -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.795..2.471 rows=1172 loops=1)
                            -> Covering index scan on t using idx_team_name  (cost=3.55 rows=33) (actual time=0.108..0.137 rows=33 loops=1)
                            -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.061..0.068 rows=36 loops=33)
                        -> Covering index lookup on g using idx_player_points (player_id=p.player_id)  (cost=0.94 rows=8) (actual time=0.003..0.004 rows=8 loops=1172)
 |
+--------------------------------------------------------------------------------------------------
```

QUERY 4:

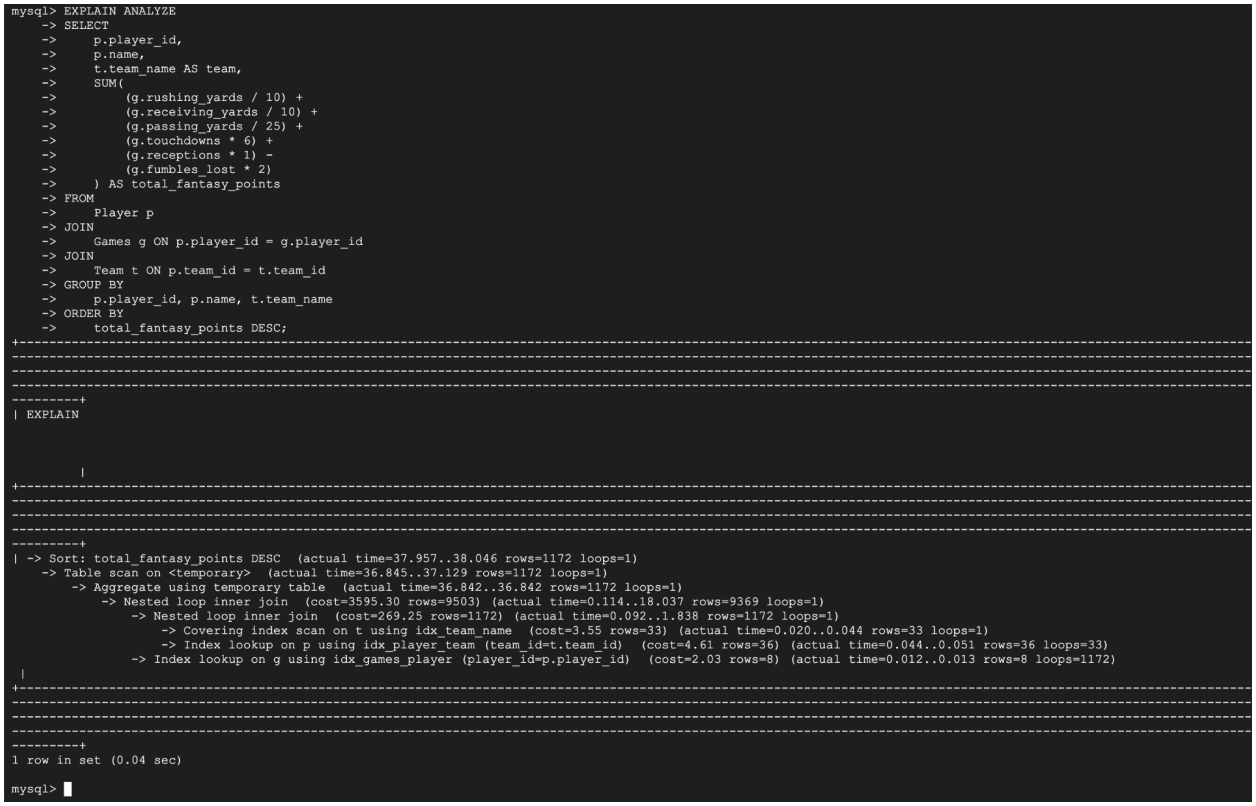
Query 4

```
mysql>
mysql> EXPLAIN ANALYZE
    -> SELECT
    ->     p.player_id,
    ->     p.name,
    ->     t.team_name AS team,
    ->     SUM(
    ->         (g.rushing_yards / 10) +
    ->         (g.receiving_yards / 10) +
    ->         (g.passing_yards / 25) +
    ->         (g.touchdowns * 6) +
    ->         (g.receptions * 1) -
    ->         (g.fumbles_lost * 2)
    ->     ) AS total_fantasy_points
    -> FROM
    ->     Player p
    -> JOIN
    ->     Games g ON p.player_id = g.player_id
    -> JOIN
    ->     Team t ON p.team_id = t.team_id
    -> GROUP BY
    ->     p.player_id, p.name, t.team_name
    -> ORDER BY
    ->     total_fantasy_points DESC;
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
---------+
| EXPLAIN



         |
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
---------+
| -> Sort: total_fantasy_points DESC  (actual time=79.750..79.897 rows=1172 loops=1)
    -> Table scan on <temporary>  (actual time=78.526..78.899 rows=1172 loops=1)
        -> Aggregate using temporary table  (actual time=78.521..78.521 rows=1172 loops=1)
            -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.207..39.025 rows=9369 loops=1)
                -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.168..3.278 rows=1172 loops=1)
                    -> Covering index scan on t using idx_team_name  (cost=3.55 rows=33) (actual time=0.055..0.131 rows=33 loops=1)
                    -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actual time=0.077..0.092 rows=36 loops=33)
                -> Index lookup on g using idx_games_player (player_id=p.player_id)  (cost=2.03 rows=8) (actual time=0.027..0.029 rows=8 loops=1172)
 |
+------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
---------+
1 row in set (0.10 sec)
```

Strategy 1: Indexing the player_id and team_id Columns in Games and Player Tables



- The nested inner join operations now have lower costs. The Index lookup on p using idx_player_team (team_id=t.team_id) also shows a relatively low cost (around 4.61). Overall, the cost is 38.046, so the sorting operation is still costly. But due to the join efficiency, there is still an improved efficiency.

Strategy 2: Composite index on player_id and team_id in the Games table will reduce the cost of accessing rows based on these attributes.
CREATE INDEX idx_games_player_team ON Games(player_id, home_team);

```
                              |
+-------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
-----------------------------+
| -> Sort: total_fantasy_points DESC  (actual time=37.428..37.512 rows=1172 loops=1)
    -> Table scan on <temporary>  (actual time=36.634..36.929 rows=1172 loops=1)
        -> Aggregate using temporary table  (actual time=36.630..36.630 rows=1172 loops=1)
            -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.135..17.938 rows=9369 loops=1)
                -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.111..1.848 rows=1172 loops=1
)
                    -> Covering index scan on t using idx_team_name  (cost=3.55 rows=33) (actual time=0.034..0
.059 rows=33 loops=1)
                    -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actua
l time=0.043..0.052 rows=36 loops=33)
                -> Index lookup on g using idx_games_player (player_id=p.player_id)  (cost=2.03 rows=8) (actua
l time=0.012..0.013 rows=8 loops=1172)
 |
+-------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
-----------------------------+
1 row in set (0.04 sec)

mysql>
```

The Games(player_id, home_team) index does not seem to work in the EXPLAIN ANALYZE output. The query still relies on the individual index idx_games_player to look for player_id. The costs are similar for idx_games_player, and idx_player_teams for joins. In the future, we realize that the individual indexes (idx_games_player and idx_player_team) will already be sufficient to optimize the joins and lookups.

Strategy 3: Indexing Columns Used in GROUP BY and ORDER BY Clauses (team_name in Team and player_id in Player)

```
+-------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------------------------------------
-----------------------------+
| -> Sort: total_fantasy_points DESC  (actual time=40.068..40.151 rows=1172 loops=1)
    -> Table scan on <temporary>  (actual time=39.301..39.575 rows=1172 loops=1)
        -> Aggregate using temporary table  (actual time=39.298..39.298 rows=1172 loops=1)
            -> Nested loop inner join  (cost=3595.30 rows=9503) (actual time=0.186..20.165 rows=9369 loops=1)
                -> Nested loop inner join  (cost=269.25 rows=1172) (actual time=0.163..2.471 rows=1172 loops=1
)
                    -> Covering index scan on t using idx_team_name  (cost=3.55 rows=33) (actual time=0.034..0
.072 rows=33 loops=1)
                    -> Index lookup on p using idx_player_team (team_id=t.team_id)  (cost=4.61 rows=36) (actua
l time=0.058..0.070 rows=36 loops=33)
                -> Index lookup on g using idx_games_player (player_id=p.player_id)  (cost=2.03 rows=8) (actua
l time=0.013..0.014 rows=8 loops=1172)
 |
```

The EXPLAIN ANALYZE output indicates a covering index scan on idx_team_team_name for team_name, which may help with sorting/grouping operations on this column. The cost for this scan appears to be low, which is beneficial. However, there is no significant reduction or substantial performance gains.

Corrections for Stage 2 are located in the updated file in the docs file under Revised_stage_2.pdf