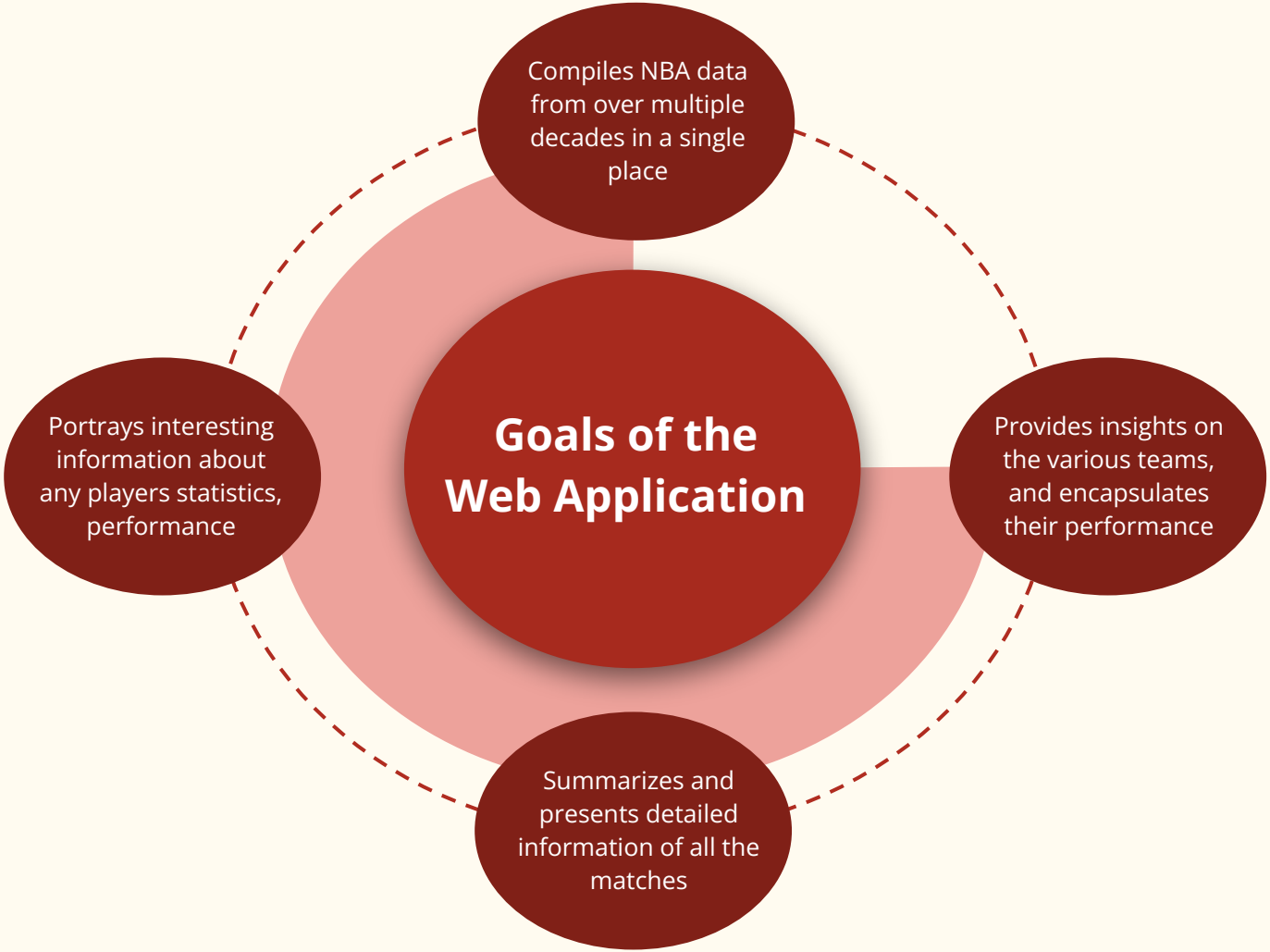
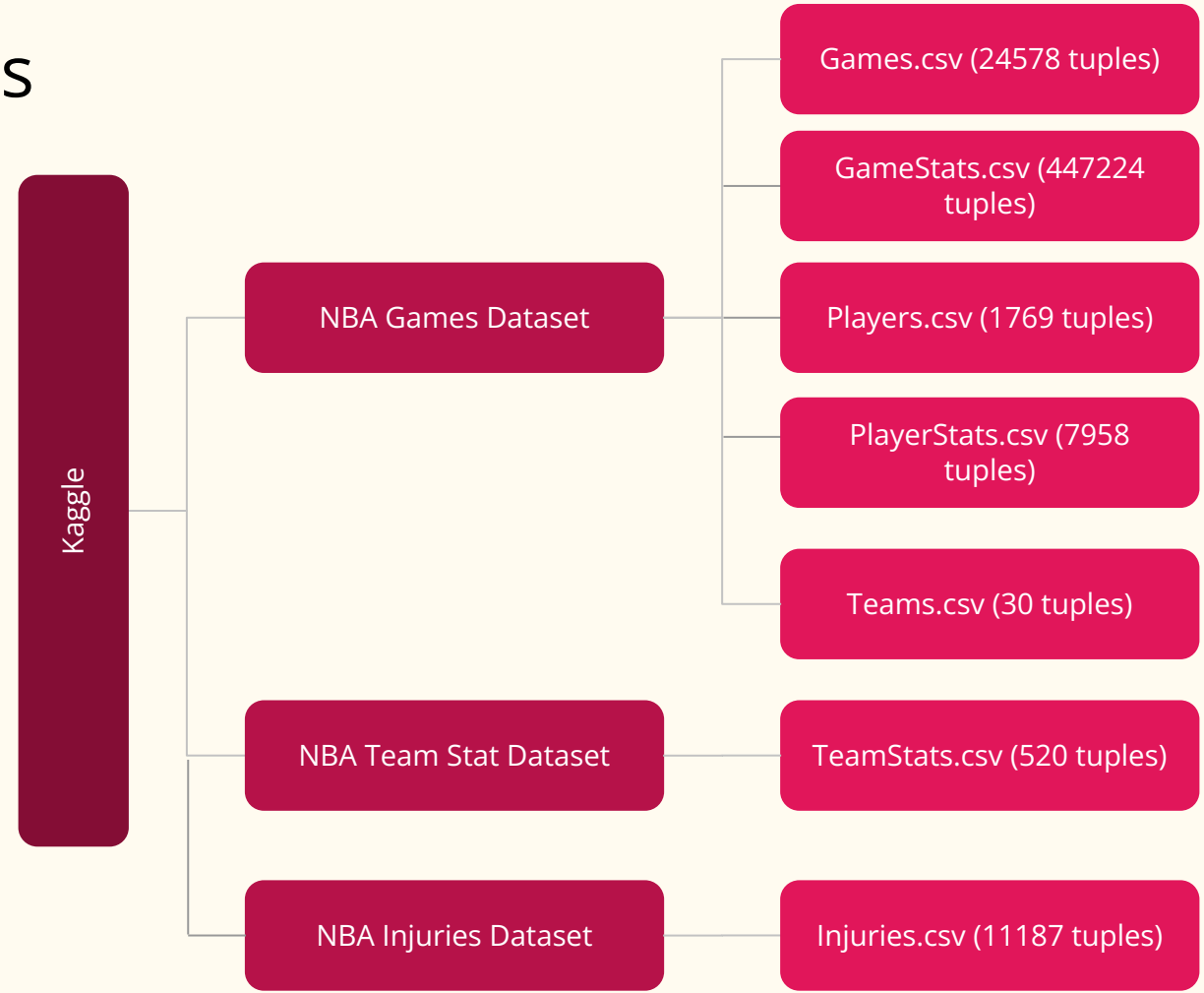
NBA Game Time

Team Members - Jay Gala, Mitali Mishra, Saurabh Raut, Vandana Miglani

Team Mentor - Hayden Hyunjoo Ji


Compiles NBA data from over multiple decades in a single place
Goals of the Web Application
Portrays interesting information about any players statistics, performance
Provides insights on the various teams, and encapsulates their performance
Summarizes and presents detailed information of all the matches

Datasets


Kaggle
NBA Games Dataset
Games.csv (24578 tuples)
GameStats.csv (447224 tuples)
Players.csv (1769 tuples)
PlayerStats.csv (7958 tuples)
Teams.csv (30 tuples)
NBA Team Stat Dataset
TeamStats.csv (520 tuples)
NBA Injuries Dataset
Injuries.csv (11187 tuples)

Cleaning of the Datasets

Dropped redundant columns like HOME_TEAM_ID and TEAM_ID_Home, etc..

Dropped NAN values

Changed the type of different columns to appropriate types. Example, changed string data of ‘Date’ column to DateTime type

Cleaned columns like ‘Season’ from original value of 1996-97 to 1996 (keeping the lower bound) for easy record linking with other tables

Renamed columns and kept the same names across relations for record linking for JOINS

Some snippets of Data Cleaning (Colab Notebooks can be found on Git)

In [ ]:
```
player_stats_df['Season'] = player_stats_df['Season'].apply(lambda x: x.split('-')[0])
```

In [ ]:
```
player_stats_df = player_stats_df.drop(['Player_Name', 'team_abbreviation'], axis=1)
```

In [ ]:
```
player_stats_df['Season'] = player_stats_df['Season'].astype(int)
```

In [ ]:
```
Game = Game.dropna(axis=0)
```

In [ ]:
```
teams_df = teams_df.rename(columns={"ABBREVIATION": "team_abbreviation"})
```

In [ ]:
```
Game['Game_Date_EST'] = pd.to_datetime(Game['Game_Date_EST'])
Game['PTS_Home'] = Game['PTS_Home'].astype(int)
Game['AST_Home'] = Game['AST_Home'].astype(int)
Game['REB_Home'] = Game['REB_Home'].astype(int)
Game['PTS_Away'] = Game['PTS_Away'].astype(int)
Game['AST_Away'] = Game['AST_Away'].astype(int)
Game['REB_Away'] = Game['REB_Away'].astype(int)
```

Normalization of the Datasets

As can be seen from the description of datasets, the data for this project was sourced from different data sources. So, we reorganized the dataset in a manner that would allow us to easily link relations from different datasets.

Example, While the Players table had PlayerID as an attribute already present in the table, the Player Injuries dataset did not. To tackle this problem, PlayerID is included in another column in both the PlayerStats table and the Player Injuries table. And, the Player Name column was dropped to avoid redundancies.

This same approach was also followed for the Teams Table. Here, we have the Team ID in the Teams table along with other attributes such as Team Name, etc.. In all other tables referencing Teams, such as Team Stats, Game Stats, Injuries, etc.. we only included the Team ID and dropped the Name columns to prevent redundancy.

Normalization of Database (Contd.)

Some of the Functional Dependencies -
Player_ID -> Player_Name
Team_ID -> Team_Name, Arena, General Manager, Affiliation, ...
Player_ID, Season -> Player_Height, Player_Weight, Gp, Pts, Reb, ... {This allows us to see season wise performance of the player}

BCNF Checking:

Conditions for BCNF are -
For every relation schema and for every functional dependency X->A, X is a superkey of R.

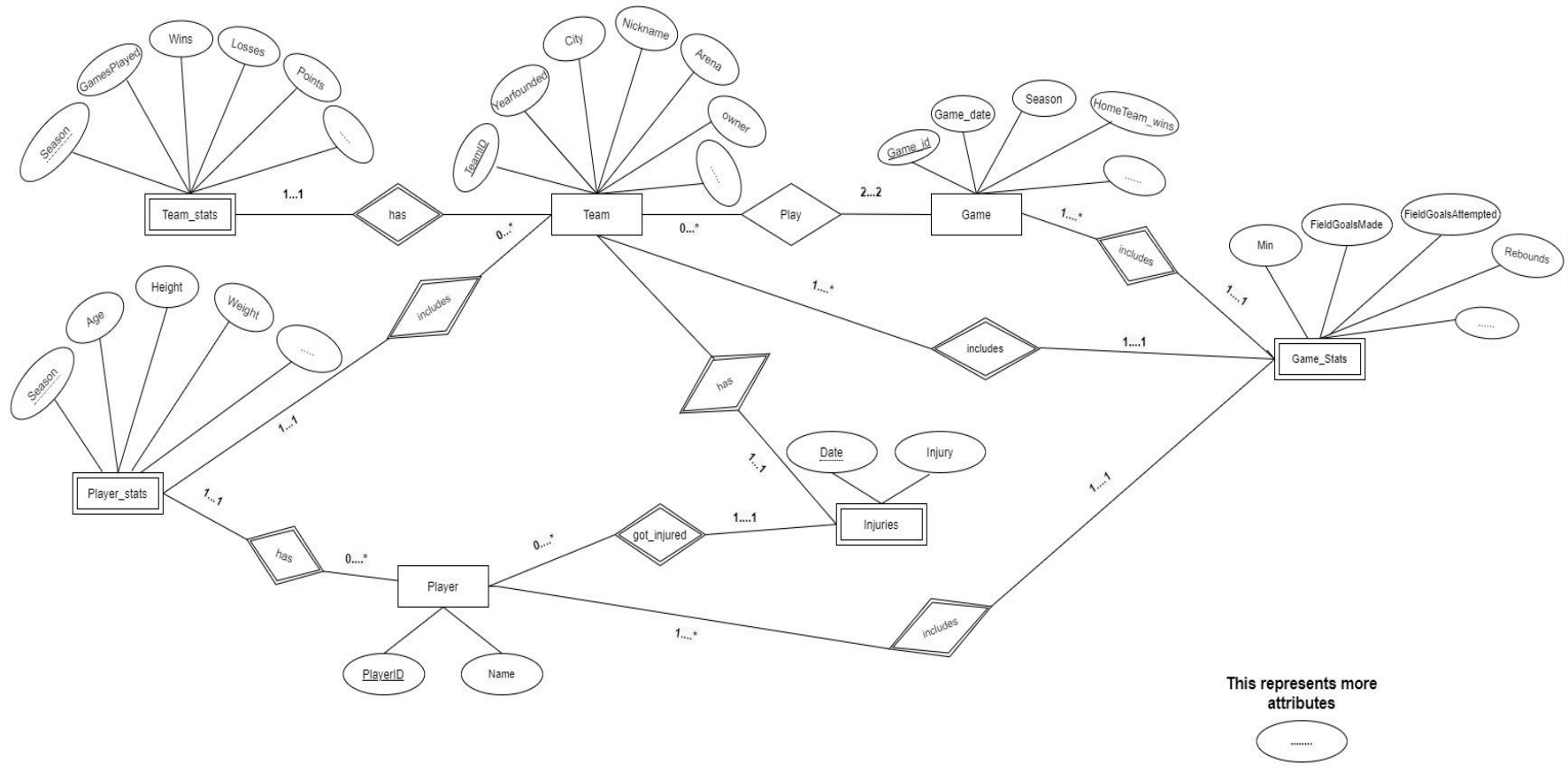
As can be seen from the reorganized schema, the tables are in BCNF. Example, in the Players Table with dependency Player_ID -> Player_Name, Player_ID is the superkey of the table. Similarly, in this functional dependency Team_ID -> Team_Name, Arena, General Manager, Affiliation, ..., Team_ID is the superkey. **So, the tables are in BCNF.**

Schema Design

Table	Purpose	Schema
Team	Information about NBA Teams	(**Team_ID**, Abbreviation, Year_Founded, Team_Name, Arena, Arena_Capacity, Owner, General_Manager, HeadCoach, DLeague_Affiliation)
Team Stats	Information about teams performance per season	(***T*eam_ID**, **Season**,GP,W, L, Win_Percentage, Min, Pts,FGM, FGA, FG_Percentage, 3PM, 3PA, 3P_Percentage, FTM, FTA, FT_Percentage, OREB, DREB, REB, AST, TOV, STL, BLK, BLKA, PF, PFD, Plus_Minus,**Team_ID FOREIGN KEY REFERENCES Team (Team_ID)**)
Players	List of players with unique ids	(**Player_ID**, Player_Name)
Player Stats		**Player_ID**, **Season**, Team_ID, Age, Player_Height, Player_weight, College, Country, Draft_Year, Draft_Round, Draft_Number, Gp, Pts, Reb, Ast, Net_Rating, Oreb_Pct, Dreb_Pct, Usg_Pct, Ts_Pct, Ast_Pct, **Player_ID FOREIGN KEY REFERENCES Player(Player_ID), Team_ID FOREIGN KEY REFERENCES Team(Team_ID)**)

Table	Purpose	Schema
Game	Information about games with points scored by participating teams	**(<u>Game_ID</u>**,Game_Date_EST, Home_Team_ID, Visitor_Team_ID, Season, PTS_Home, FG_PCT_Home, FT_PCT_Home, FG3_PCT_Home, AST_Home, REB_Home, PTS_Away, FG_PCT_Away, FT_PCT_Away, FG3_PCT_Away, AST_Away, REB_Away, Home_Team_Wins, **Home_Team_ID FOREIGN KEY REFERENCES Team(Team_ID), Visitor_Team_ID FOREIGN KEY REFERENCES Team(Team_ID)**)
Game Stats	Information about points scored by each player of each team in a game	(**<u>Game_ID,Team_ID, Player_ID</u>**, Min, FGM, FGA, FG_PCT, FG3M, FG3A, FG3_PCT, FTM, FTA, FT_PCT, OREB, DREB, REB, AST, STL, BLK, TO, PF, PTS, Plus_Minus,Game_ID **FOREIGN KEY REFERENCES Game(Game_ID), Team_ID FOREIGN KEY REFERENCES Team(Team_ID), Player_ID FOREIGN KEY REFERENCES Player(Player_ID)**)
Injuries	Information about injuries of players	(**<u>Player_ID</u>**, Date, Team_ID, Injury_Info, **Team_ID FOREIGN KEY REFERENCES Team (Team_ID)**)

ER Diagram


Team_stats
Season
GamesPlayed
Wins
Losses
Points
.....
1...1
has
0...*
Team
TeamID
Yearfounded
City
Nickname
Arena
owner
.....
0...*
Play
2...2
Game
Game_id
Game_date
Season
HomeTeam_wins
.....
1....*
includes
1....1
Game_Stats
Min
FieldGoalsMade
FieldGoalsAttempted
Rebounds
.....
1....*
includes
1....1
Player_stats
Season
Age
Height
Weight
.....
includes
1...1
1...1
has
0....*
Player
PlayerID
Name
0...*
got_injured
1....1
Injuries
Date
Injury
1...1
has
1....*
includes
This represents more attributes
.....

DEMO

Complex Query 1 -

Given a particular player, we have to choose a team and this query will show how that particular player has performed against that team.

For example, in the above mentioned query, we will show how LeBron James has performed against the Boston Celtics over the years - what has his record been against them. Moreover, we are sorting based on recency of the games.

This feature is used significantly by many news reporters and match preview specialists to create X-factors, key players for a particular game, or even player vs player battles.

Optimizations:

1. Used Inner Joins instead of Cartesian Products
2. Projecting only the required attributes and moving it as inside as possible
3. Created 2 indexes on GameStats - one on Game_ID and the other on Team_ID

```
WITH TeamID AS ( SELECT Team_ID
        FROM Teams
        WHERE Team_Name = 'Boston Celtics' ),
        GameInfo AS ( SELECT Game_ID, Game_Date_EST
        FROM Game, TeamID
        WHERE Home_Team_ID = TeamID.Team_ID
          or Visitor_Team_ID = TeamID.Team_ID )
    SELECT GameInfo.Game_Date_EST, GameStats.Min,
        GameStats.PTS,
          GameStats.FGM, GameStats.FGA, GameStats.FG_PCT,
          GameStats.FG3_PCT, GameStats.FT_PCT,
          GameStats.FG3M, GameStats.FTM,
          GameStats.FG3A, GameStats.FTA,
          GameStats.REB,
          GameStats.AST, GameStats.STL, GameStats.BLK,
GameStats.Turnover
    FROM GameStats
      JOIN
      TeamID
        ON GameStats.Team_ID != TeamID.Team_ID
      JOIN
      GameInfo
        ON GameInfo.Game_ID = GameStats.Game_ID
     WHERE GameStats.Player_ID = 2544
    ORDER BY GameInfo.Game_Date_EST DESC;
```

Complex Query 2 -

Given a particular player id,this query will result in identifying the score statistics of that player while he played for a team for more than two seasons and when that team won the games that it played.

For example, a given player say LeBron James played for a team Cleveland Cavaliers for 6 seasons and scored 50% of the points for most games whenever the team won while he was in the team.

Optimizations:

1. Used Inner Joins instead of Cartesian Products
2. Projecting only the required attributes and moving it as inside as possible
3. Redefined where clause having nested queries
4. Created index on GameStats using Game_ID

```
With playerinfo (pid,pteam_id,num_seasons) as
(select p1.Player_ID,p1.Team_ID,count(p1.Season) as num_seasons
from PlayerStats p1
where p1.Player_ID=  {player_id}
group by p1.Team_ID
having num_seasons > 2),

gameinfo (gameid,teamid,points,season) as (
select g.Game_ID,g.Home_Team_ID as team,g.PTS_Home as points,Season
from Game g join playerinfo pi on (pi.pteam_id = g.Home_Team_ID)
where g.Home_Team_Wins = 1
UNION
select  g.Game_ID,g.Visitor_Team_ID as team,g.PTS_Away as points,Season
from Game g join playerinfo pi on (pi.pteam_id = g.Visitor_Team_ID)
where g.Home_Team_Wins <> 1),

player_gameinfo(Game_ID,Team_ID,pid,Min,FGM,F3GM,FTM,REB,AST,PF,PTs,perc,seas
on) as (
Select gs.Game_ID, gs.Team_ID, pi.pid, gs.Min, gs.FGM, gs.FG3M, gs.FTM, gs.REB,
gs.AST, gs.PF, gs.PTS,round((gs.PTS/gi.points)*100,2) as percentage,gi.season
from GameStats gs
            join gameinfo gi on (gi.gameid = gs.Game_ID  and gi.teamid =
        gs.Team_ID)
          join playerinfo pi on (pi.pid = gs.Player_ID)
order by percentage DESC)

Select pgi.Game_ID, pgi.Team_ID, t.Team_Name, pgi.pid, pgi.Min, pgi.FGM, pgi.F3GM,
pgi.FTM, pgi.REB,pgi.AST,pgi.PF,pgi.PTs,pgi.perc,pgi.season
from player_gameinfo pgi join Teams t on pgi.Team_ID = t.Team_ID
order by perc DESC;
```

Complex Query 3 -

Description -

Given a Team ID, we want to compute the star players along with the number of points scored in their best performance.

For this computation, we first select the seasons that this team played in. Using this subresult, we can compute the games this team played in (where it could either be a home team/visitor team). From here, we can find out which players from the team played in which game and extract the best game-wise performance among all players. From here, we can then find the best performance across all seasons among these top players and display only the top 5.

Optimizations:

1. Inner Joins as opposed to Cartesian Products
2. Extracting only the seasons for particular team very early on
3. Projecting only the required attributes from one sub result to another
4. Indexes on Game and GameStats Tables

```
WITH TeamIDResults (Team_ID, Season) AS (
            SELECT TS.Team_ID, TS.Season
            from TeamStats TS WHERE TS.Team_ID = ${Team_Id}
),
GameIDResults (Team_ID, Season, Game_ID) AS (
       SELECT T.Team_ID, T.Season, G.Game_ID
       From
       TeamIDResults T JOIN Game G ON G.Home_Team_ID = T.Team_ID
       UNION
       SELECT T.Team_ID, T.Season, G.Game_ID
       From
       TeamIDResults T JOIN Game G ON G.Visitor_Team_ID = T.Team_ID
),
PlayerResults AS (
   SELECT GS.Player_ID, GResults.Team_ID, GResults.Season,
GResults.Game_ID, GS.PTS
   FROM GameIDResults GResults JOIN GameStats GS ON GResults.Game_ID
= GS.Game_ID
   AND GResults.Team_ID = GS.Team_ID
),
TopPlayers AS (
   SELECT MAX(PTS) AS PtsScored, PR.Player_ID, Season, Game_ID
   FROM PlayerResults PR
   GROUP BY Season, Game_ID
   ORDER BY PTS DESC
)
SELECT MAX(PtsScored) AS MaxPts, TP.Player_ID, P.Player_Name,
TP.Game_ID, Season FROM TopPlayers TP JOIN Players P
ON P.Player_ID = TP.Player_ID
GROUP BY Season
ORDER BY MaxPts DESC
LIMIT 5
```

Complex Query 4 -

Given a particular Game and Team this query finds the season average performance of each team player that played in that particular Game.

So let's say if we select a Game G to find how the Home Team Players have performed in G, this query will allow us to also see and compare how the same Players have performed on an average in this season

Optimizations:

1. Created index SeasonFinder on Season column of Game table to find seasons faster
2. Projected only required attributes
3. Used Inner joins instead of Cartesian products

```
WITH TeamPlayerIDS AS (
    SELECT P2.Player_ID
    FROM (Players P2 JOIN GameStats GS2 ON P2.Player_ID =
GS2.Player_ID)
    WHERE GS2.Game_ID = ${req.params.GameID}  AND GS2.Team_ID =
${req.params.Team_ID}
)
SELECT P.Player_Name,
ROUND(AVG(GS.PTS),2) as AVG_Points_Scored_by_the_Player,
ROUND(AVG(GS.Min),2) as AVG_Minutes_Played,
ROUND(AVG(GS.FGM),2) as AVG_Field_Goals_Made, ROUND(AVG(GS.FGA),2)
as AVG_Field_Goals_Attempted,ROUND(AVG(GS.FG_PCT),2) as
AVG_Field_Goal_Percentage,ROUND(AVG(GS.FG3M),2) as
AVG_Three_Pointers_Made, ROUND(AVG(GS.FG3A),2) as
AVG_Three_Pointers_Attempted,ROUND(AVG(GS.FG3_PCT),2) as
AVG_Three_Point_Percentage, ROUND(AVG(GS.FTM),2) as
AVG_Free_Throws_Made,ROUND(AVG(GS.FTA),2) as
AVG_Free_Throws_Attempted, ROUND(AVG(GS.FT_PCT),2) as
AVG_Free_Throw_Percentage,ROUND(AVG(GS.OREB),2) as
AVG_Offensive_Rebounds, ROUND(AVG(GS.DREB),2) as
AVG_Defensive_Rebounds,ROUND(AVG(GS.REB),2) as AVG_Rebounds,
ROUND(AVG(GS.AST),2) as AVG_Assists,ROUND(AVG(GS.STL),2) as
AVG_Steals, ROUND(AVG(GS.BLK),2) as
AVG_Blocked_Shots,ROUND(AVG(GS.PF),2) as AVG_Personal_Foul,
ROUND(AVG(GS.PLUS_MINUS),2) as AVG_PLUS_MINUS
FROM (GameStats GS JOIN Game G on GS.Game_ID = G.Game_ID)
JOIN Players P on P.Player_ID = GS.Player_ID
JOIN Teams T on T.Team_ID = GS.Team_ID
JOIN TeamStats TS on T.Team_ID = TS.Team_ID
WHERE G.Season IN (SELECT G2.Season FROM Game G2 WHERE G2.Game_ID =
${req.params.GameID})
AND P.Player_ID IN (SELECT Player_ID FROM TeamPlayerIDS)
GROUP BY P.Player_ID, P.Player_Name;
```

Performance

Query Description	Original	Optimised
Player Performance vs a particular opponent	2s 62ms	676ms
Player Performance in a team for more than 2 seasons	3s 362ms	633ms
Finding the star performers of a particular team across different seasons, alongwith their best performance (in points scored)	2s 499ms	749ms
Average performance of the Game players in that Particular season	3s 67ms	813ms

Technical Challenges

01 **Data Cleaning and Preprocessing**

- Raw data had to be cleaned using packages like numpy and pandas
- Inconsistent forms of data - eg: minutes played, date
- Enforcing key constraints - by having left joins

02 **Large Dataset**

- Gave multiple errors while loading dataset into the AWS database

03 **Complex Queries**

- Tables were very self sufficient - contained all necessary information within themselves - making joins unrequired

THANK YOU!