PROJECT TITLE

Database Systems Project- NBA Games

Project Goal:

This project aims to import and evaluate the NBA games dataset using PostgreSQL in PgAdmin. The NBA games dataset on Kaggle at this link has been used:
https://www.kaggle.com/datasets/nathanlauga/nba-games?resource=download

This dataset contains data about NBA (National Basketball Association) matches from the season 2004 to season 2022. It also contains information on the respective NBA players and their teams, statistics, maximum and minimum points scored by the players/teams, etc.

The dataset consists of 5 tables:

1. **Games:** Contain all games from the 2004 season to the 2022 season with the date, teams, and some details like the number of points, etc.
2. **Games Details:** Contain the records and details of the game's dataset, all statistics of NBA players who participated in the given seasons and for a given game.
3. **Players:** Contains names of all the players and in which team they are present in the most recent season available in the dataset.
4. **Ranking:** Provides the list of everyday ranking of all the teams. It is dynamic and trends to change every day based on the number of games won/lost.
5. **Teams:** Contains various information with respect to the name of the team, the year in which the team was founded, owner, manager, arena capacity, etc.

ATTACHED FILES

- NBA_DATASET
- DDL statements: DDL_STATEMENTS.txt,
- DML statements: DML_STATEMENTS.txt

- QUERIES
- Index.zip: Indexing
- VERBOSE: Verbose.txt
- OTHER_FILES
- PROJECT_PROPOSAL

DATASET

This dataset contains data about NBA (National Basketball Association) matches from the season 2004 to season 2022. It also, contains information on the respective NBA players and their teams, statistics, maximum and minimum points scored by the players/teams, etc.

It has been sourced from Kaggle.

https://www.kaggle.com/datasets/nathanlauga/nba-games?resource=download The dataset consists of 5 relations:

1. **Games:** Contain all games from the 2004 season to the 2022 season with the date, teams, and some details like the number of points, etc.
2. **Games Details:** Contain the records and details of the game's dataset, all statistics of NBA players who participated in the given seasons and for a given game.
3. **Players:** Contains names of all the players and in which team they are present in the most recent season available in the dataset.
4. **Ranking:** Provides the list of everyday ranking of all the teams. It is dynamic and trends to change every day based on the number of games won/lost.
5. **Teams:** Contains various information with respect to the name of the team, the year in which the team was founded, owner, manager, arena capacity, etc. Below is the snapshot of the relation 'Players'

```
Jeff Green,1610612762,201145,2019
Dante Exum,1610612762,203957,2019
Emmanuel Mudiay,1610612762,1626144,2019
Georges Niang,1610612762,1627777,2019
Tony Bradley,1610612762,1628396,2019
Nigel Williams-Goss,1610612762,1628430,2019
Tobias Harris,1610612755,202699,2019
Al Horford,1610612755,201143,2019
```

DATASET TRANSFORMATION:

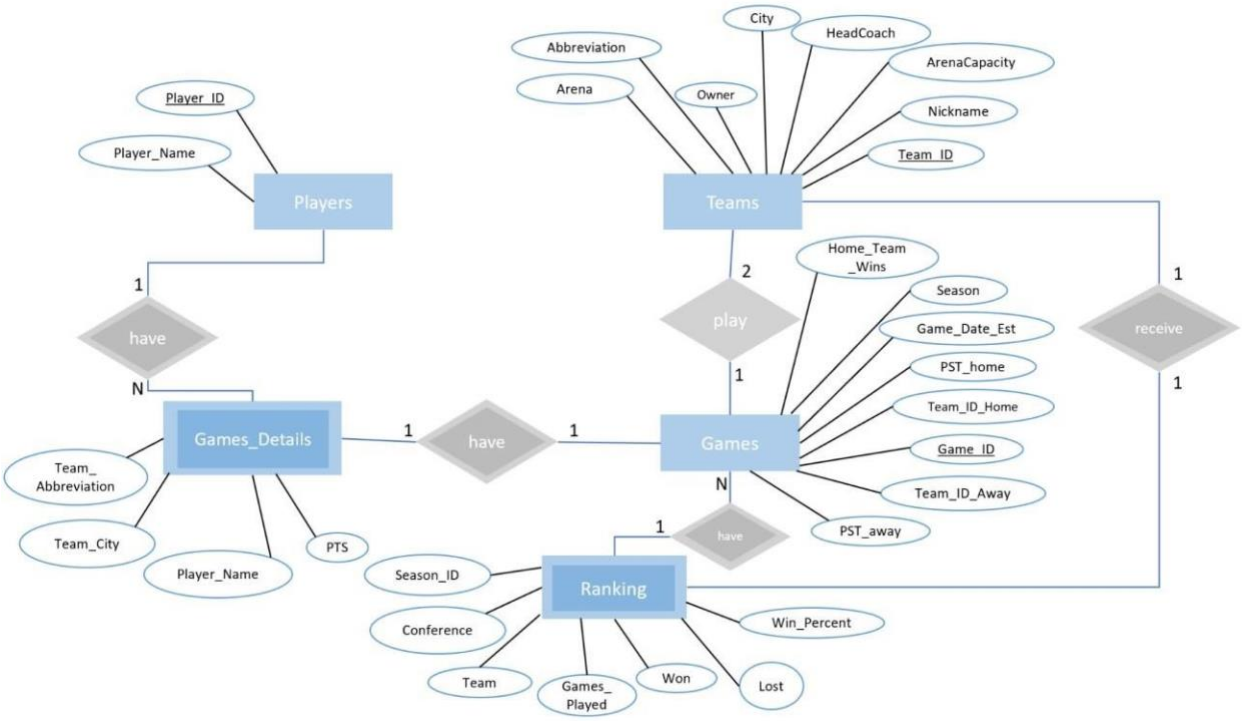
Used Excel functions to identify and correct errors, inconsistencies, and missing values

● Removed duplicate entries using Excel's built-in Remove Duplicates command

- Imported and structured data from text files using Excel's Text to Columns function
- Changed the character encoding format to UTF-8 to ensure compatibility with analysis tools
- Ensured data accuracy, completeness, and representation for analysis purposes
- This revised version highlights the main steps taken to clean and prepare the data for analysis, while maintaining clarity and organization.

CONCEPTUAL DESIGN (ER DIAGRAM)

The ER diagram below provides a faithful, complete and unambiguous specification of the data stored in the NBA Games Dataset;



DATABASE SCHEMA:

Below is the table creation schema:

BELOW IS THE SCHEMA FOR:

TEAMS TABLE
GAMES TABLE
PLAYER TABLE
RANKING TABLE
GAMES_DETAILS TABLE

```
CREATE DATABASE "CIS556NBAproject"

    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;

 COMMENT ON DATABASE "CIS556NBAproject"
    IS 'CIS556 Winter 2023 term NBA project';
DROP TABLE IF EXISTS Teams CASCADE;
DROP TABLE IF EXISTS Player CASCADE;
DROP TABLE IF EXISTS Ranking CASCADE;
DROP TABLE IF EXISTS Games CASCADE;
DROP TABLE IF EXISTS Games_Details CASCADE;

CREATE TABLE Teams (
    TEAM_ID INT PRIMARY KEY
          , ABBREVIATION VARCHAR (10)
          , NICKNAME VARCHAR (15)
          , CITY VARCHAR (15)
          , ARENA VARCHAR (50)
          , ARENACAPACITY INT
          , OWNER VARCHAR (50)
          , HEADCOACH VARCHAR (50)
);


CREATE TABLE GAMES (
       GAME_ID INT PRIMARY KEY
          , GAME_DATE_EST DATE
          , HOME_TEAM_ID INT REFERENCES Teams (TEAM_ID)
          , VISITOR_TEAM_ID INT REFERENCES Teams (TEAM_ID)
```

```
        , SEASON INT
        , TEAM_ID_HOME INT
        , PTS_HOME INT
        , TEAM_ID_AWAY INT
        , PTS_AWAY INT
        , HOME_TEAM_WINS BOOLEAN
);


CREATE TABLE Player (
      PLAYER_ID INT PRIMARY KEY
        , PLAYER_NAME VARCHAR (50)
);

CREATE TABLE RANKING (
   TEAM_ID INT REFERENCES TEAMS(TEAM_ID)
        , SEASON_ID INT
        , CONFERENCE VARCHAR (15)
        , TEAM VARCHAR (15)
        , GAMES_PLAYED INT
        , WON INT
        , LOST INT
        , WIN_PERCENT FLOAT
);

CREATE TABLE GAMES_DETAILS (
         GAME_ID INT REFERENCES GAMES(GAME_ID)
        , TEAM_ID INT REFERENCES TEAM(TEAM_ID)
        , TEAM_ABBREVIATION VARCHAR (15)
        , TEAM_CITY VARCHAR (50)
        , PLAYER_ID INT REFERENCES PLAYER(PLAYER_ID)
        , PLAYER_NAME VARCHAR (50)
        , PTS INT
);
```

DML STATEMENTS:

These statements are stored in the tables_import_query.txt file.

```
COPY Teams (
            TEAM_ID
           , ABBREVIATION
           , NICKNAME
           , CITY
           , ARENA
           , ARENACAPACITY
           , OWNER
           , HEADCOACH
)
FROM 'C:\Users\DELL\Documents\Semester1\CIS556\Project\NBA_DATASET\nba_teams.csv'
DELIMITER ','
CSV HEADER;


COPY Player (
            PLAYER_ID
           , PLAYER_NAME
           , TEAM_ID
           , SEASON
)
FROM 'C:\Users\DELL\Documents\Semester1\CIS556\Project\NBA_DATASET\nba_players.csv'
DELIMITER ','
CSV HEADER;


COPY GAMES (
            GAME_ID
           , GAME_DATE_EST
           , HOME_TEAM_ID
           , VISITOR_TEAM_ID
           , SEASON
           , TEAM_ID_HOME
           , PTS_HOME
           , TEAM_ID_AWAY
           , PTS_AWAY
           , HOME_TEAM_WINS
)
```

```
FROM 'C:\Users\DELL\Documents\Semester1\CIS556\Project\NBA_DATASET\nba_games.csv'
DELIMITER ','
CSV HEADER;


COPY RANKING (
            TEAM_ID
           , SEASON_ID
           , CONFERENCE
           , TEAM
           , GAMES_PLAYED
           , WON
           , LOST
           , WIN_PERCENT
)
FROM 'C:\Users\DELL\Documents\Semester1\CIS556\Project\NBA_DATASET\nba_ranking.csv'
DELIMITER ','
CSV HEADER;


COPY GAMES_DETAILS (
            GAME_ID
           , TEAM_ID
           , TEAM_ABBREVIATION
           , TEAM_CITY
           , PLAYER_ID
           , PLAYER_NAME
           , PTS
)
FROM
'C:\Users\DELL\Documents\Semester1\CIS556\Project\NBA_DATASET\nba_games_details.csv'
DELIMITER ','
CSV HEADER;
```

METHODOLOGY

Query 1 has been used to verify the performance of PostgreSQL indexes. It is defined as follows:

```
SELECT GD.PLAYER_NAME

, GD.PTS

, G.GAME_DATE_EST

, CASE WHEN G.HOME_TEAM_ID = GD.TEAM_ID THEN T1.CITY ELSE T2.CITY END AS
OPPONENT_CITY

, CASE WHEN G.HOME_TEAM_ID = GD.TEAM_ID THEN T1.ABBREVIATION ELSE
T2.ABBREVIATION
END AS OPPONENT_ABBREVIATION

FROM GAMES_DETAILS GD

JOIN GAMES G

ON GD.GAME_ID = G.GAME_ID

JOIN TEAMS T1

ON G.HOME_TEAM_ID = T1.TEAM_ID

JOIN TEAMS T2

ON G.VISITOR_TEAM_ID = T2.TEAM_ID

GROUP BY GD.PLAYER_NAME, GD.PTS, G.GAME_DATE_EST, G.HOME_TEAM_ID, GD.TEAM_ID,
T1.CITY, T2.CITY, T1.ABBREVIATION, T2.ABBREVIATION

HAVING GD.PTS > (SELECT MIN(PTS) FROM GAMES_DETAILS)

ORDER BY GD.PTS DESC

LIMIT 10;
```

The command EXPLAIN ANALYZE <query> was used to test its execution time

Q3 was run in three different situations:

1. Before any statistics collection or indexing

2. With statistics but without indexes

3. With both statistics and indexes

The following commands were used to collect the table statistics:

ANALYSE VERBOSE GAMES_DETAILS

INFO: analyzing "public.games_details"

INFO: "games_details": scanned 6228 of 6228 pages, containing 668628 live rows and 0 dead rows; 30000 rows in the sample, 668628 estimated total rows

ANALYZE

The query returned successfully in 593 msec.

ANALYSE VERBOSE GAMES

INFO: analyzing "public.games"

INFO: "games": scanned 222 of 222 pages, containing 26622 live rows and 0 dead rows; 26622 rows in the sample, 26622 estimated total rows

ANALYZE

The query returned successfully in 214 msec.

ANALYSE VERBOSE PLAYER

INFO: analyzing "public.player"

INFO: "player": scanned 17 of 17 pages, containing 2687 live rows and 0 dead rows; 2687 rows in sample, 2687 estimated total rows

ANALYZE

Query returned successfully in 50 msec.

ANALYSE VERBOSE RANKING

INFO: analyzing "public.ranking"

INFO: "ranking": scanned 2585 of 2585 pages, containing 210342 live rows and 0 dead rows; 30000 rows in sample, 210342 estimated total rows

ANALYZE

Query returned successfully in 387 msec.

ANALYSE VERBOSE TEAMS

INFO: analyzing "public.teams"

INFO: "teams": scanned 1 of 1 page, containing 30 live rows and 0 dead rows; 30 rows in sample, 30 estimated total rows

ANALYZE

Query returned successfully in 45 msec.

DATABASE INDEXING

Index on the ARENA column

CREATE INDEX idx_teams_arena ON Teams (ARENA);

Index on the HEADCOACH column

CREATE INDEX idx_teams_headcoach ON Teams (HEADCOACH);

Index on both ABBREVIATION and CITY columns

CREATE INDEX idx_teams_abbrev_city ON Teams (ABBREVIATION, CITY);

Index on the PLAYER_NAME column, with case-insensitive search support

CREATE INDEX idx_player_playername_ci ON Player (lower (PLAYER_NAME));

Index on the PLAYER_ID column, with DESC ordering for better performance on large tables

CREATE INDEX idx_player_playerid_desc ON Player (PLAYER_ID DESC);

Index on both SEASON_ID and CONFERENCE columns

CREATE INDEX idx_ranking_season_conf ON Ranking (SEASON_ID, CONFERENCE);

Index on both SEASON_ID and TEAM columns

CREATE INDEX idx_ranking_season_team ON Ranking (SEASON_ID, TEAM);

Index on both HOME_TEAM_ID and VISITOR_TEAM_ID columns

CREATE INDEX idx_games_teams ON Games (HOME_TEAM_ID, VISITOR_TEAM_ID); **Index on the PTS_HOME column, with DESC ordering**

CREATE INDEX idx_games_ptshome_desc ON Games (PTS_HOME DESC);

Index on both GAME_ID and TEAM_ID columns

CREATE INDEX idx_gamesdetails_game_team ON Games_Details (GAME_ID, TEAM_ID);

Index on both TEAM_ABBREVIATION and TEAM_CITY columns

CREATE INDEX idx_gamesdetails_teamabbr_city ON Games_Details (TEAM_ABBREVIATION, TEAM_CITY);

Index on the PTS column, with DESC ordering

CREATE INDEX idx_gamesdetails_pts_desc ON Games_Details (PTS DESC);

Index on the OWNER column

CREATE INDEX idx_teams_owner ON Teams (OWNER);

Index on the PLAYER_NAME column with prefix search support

CREATE INDEX idx_player_playername_prefix ON Player (PLAYER_NAME);

Index on the LOST column

CREATE INDEX idx_ranking_lost ON Ranking (LOST);

Index on GAME_DATE_EST, HOME_TEAM_ID, VISITOR_TEAM_ID columns

CREATE INDEX idx_games_team_date ON Games (GAME_DATE_EST, HOME_TEAM_ID, VISITOR_TEAM_ID);

Index on SEASON_ID and CONFERENCE columns

CREATE INDEX idx_ranking_season_conf ON Ranking (SEASON_ID, CONFERENCE);

Index on SEASON_ID, CONFERENCE and WIN_PERCENT columns

CREATE INDEX idx_ranking_season_conf_winpct ON Ranking (SEASON_ID, CONFERENCE, WIN_PERCENT);

Index on GAME_ID, TEAM_ID and PTS columns

CREATE INDEX idx_games_details_game_team_pts ON Games_Details (GAME_ID, TEAM_ID, PTS);

Index on HOME_TEAM_ID column

CREATE INDEX idx_games_season_home ON Games (HOME_TEAM_ID) WHERE SEASON = 2022;

Index on PLAYER_ID column with a filter condition

CREATE INDEX idx_playerID_conf_pts ON Player (PLAYER_ID) WHERE player_ID >= 500;

Index on PLAYER_ID, PTS, GAME_ID

CREATE INDEX idx_gd_player_pts_date ON Games_Details (PLAYER_ID, PTS, GAME_ID) WHERE GAME_ID BETWEEN '11200005' AND '22200477';

Index on HOME_TEAM_ID, VISITOR_TEAM_ID and HOME_TEAM_WINS columns

CREATE INDEX idx_games_teams_wins ON Games (HOME_TEAM_ID, VISITOR_TEAM_ID, HOME_TEAM_WINS);

Index on ABBREVIATION column

CREATE INDEX idx_teams_abbreviation ON Teams (ABBREVIATION);

Index on the NICKNAME column

CREATE INDEX idx_teams_nickname ON Teams (NICKNAME);

Index on the CITY column

CREATE INDEX idx_teams_city ON Teams (CITY);

Index on the TEAM_ID column

CREATE INDEX idx_ranking_teamid ON Ranking (TEAM_ID);

Index on the SEASON_ID column

CREATE INDEX idx_ranking_seasonid ON Ranking (SEASON_ID);

Index on the CONFERENCE column

CREATE INDEX idx_ranking_conference ON Ranking (CONFERENCE);

Index on the TEAM column

CREATE INDEX idx_ranking_team ON Ranking (TEAM);

Index on the VISITOR_TEAM_ID column

CREATE INDEX idx_games_visitorteamid ON Games (VISITOR_TEAM_ID); **Index on the SEASON column**

CREATE INDEX idx_games_season ON Games (SEASON);

Index on the GAME_DATE_EST column

CREATE INDEX idx_games_gamedateest ON Games (GAME_DATE_EST);

Index on the GAME_ID column

CREATE INDEX idx_gamesdetails_gameid ON Games_Details (GAME_ID);

Index on the TEAM_ID column

CREATE INDEX idx_gamesdetails_teamid ON Games_Details (TEAM_ID);

Index on the PLAYER_ID column

CREATE INDEX idx_gamesdetails_playerid ON Games_Details (PLAYER_ID);

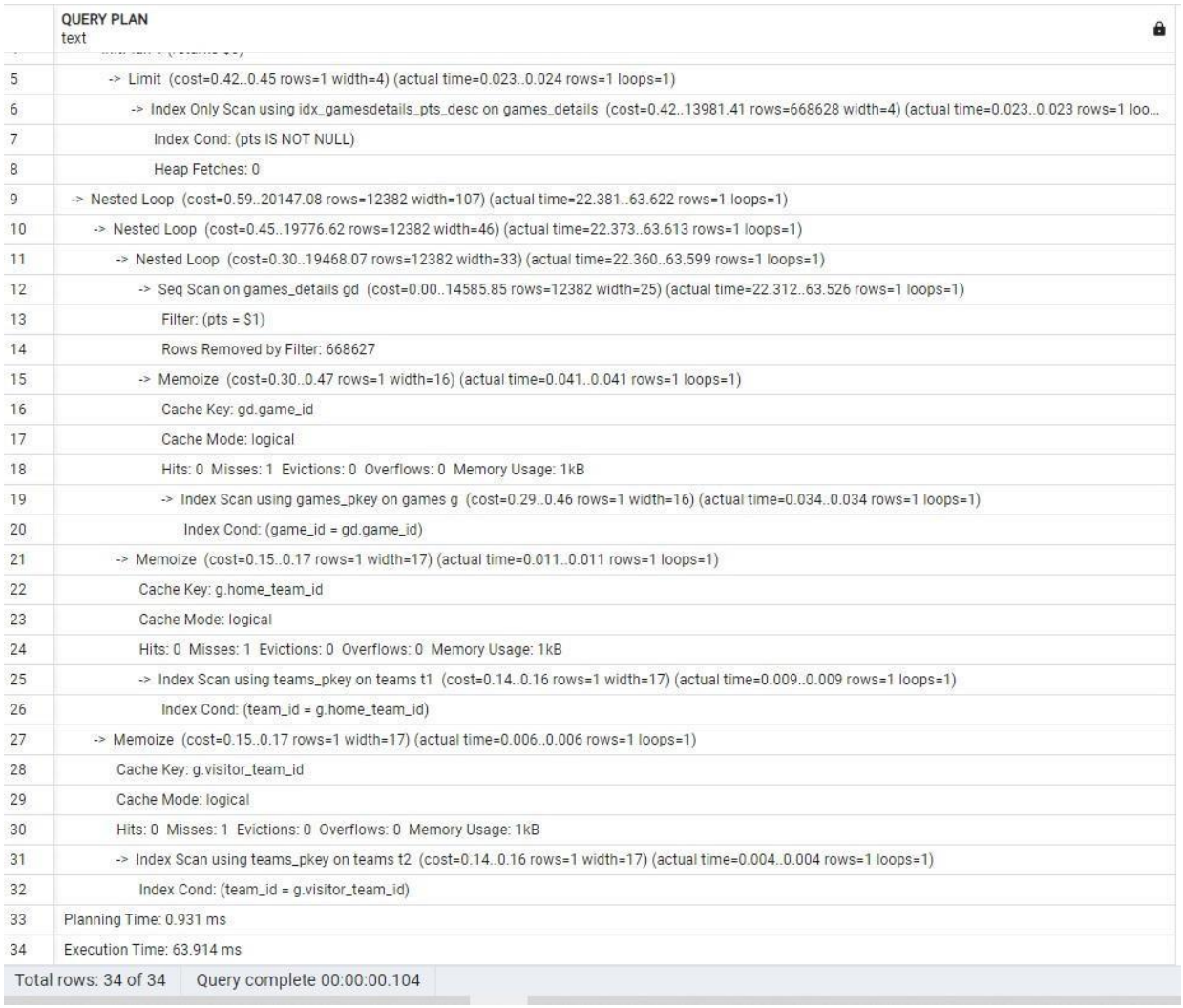
Index on the PTS column

CREATE INDEX idx_gamesdetails_pts ON Games_Details (PTS);

BENCHMARKS

This section shows the observed performance for each query execution and the query plans generated by the optimizer. Each query is run N times and found that the query execution after indexing is much faster and more efficient.

QUERY PLAN 1 BEFORE INDEXING



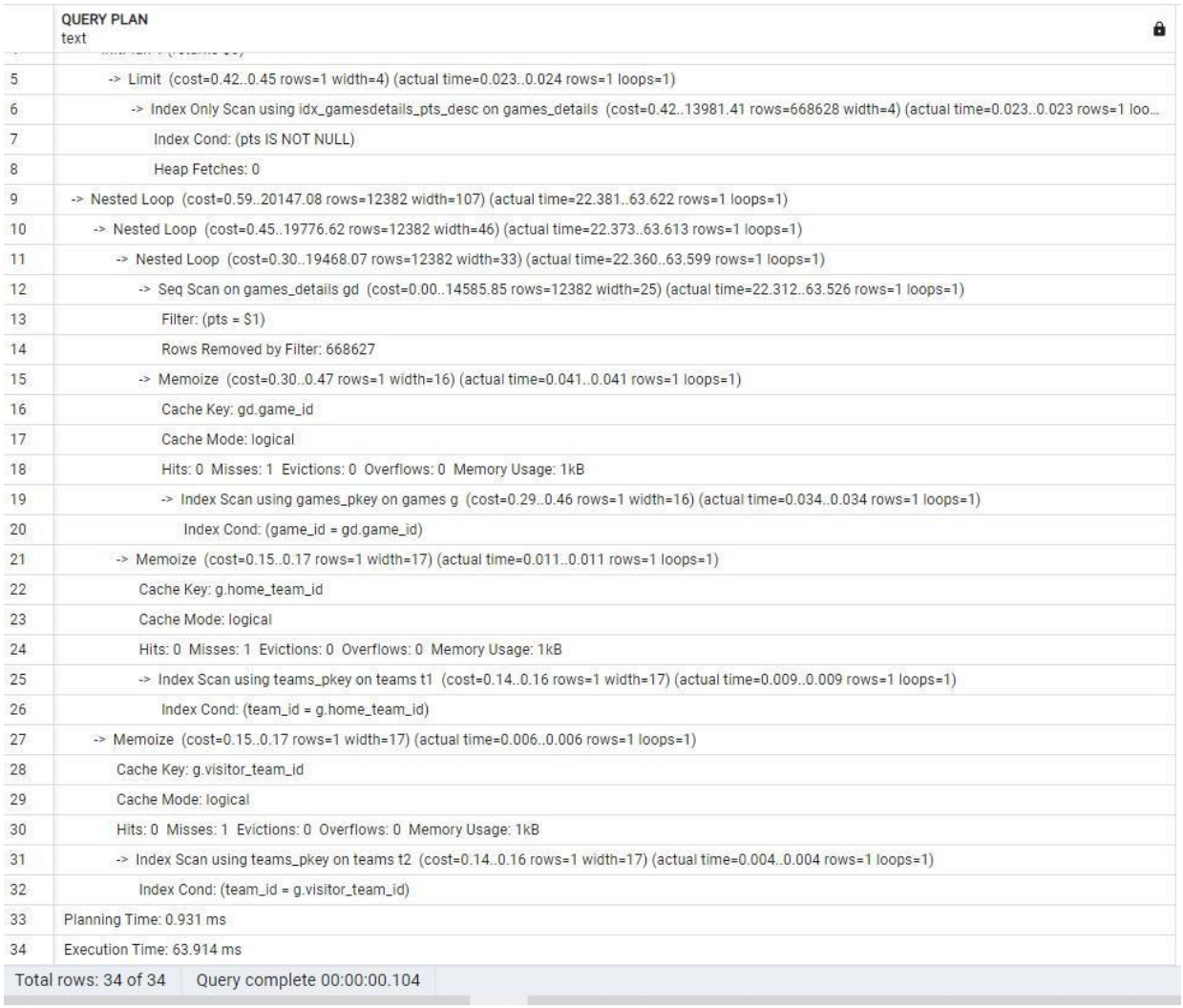
	QUERY PLAN text
5	-> Limit (cost=0.42..0.45 rows=1 width=4) (actual time=0.023..0.024 rows=1 loops=1)
6	-> Index Only Scan using idx_gamesdetails_pts_desc on games_details (cost=0.42..13981.41 rows=668628 width=4) (actual time=0.023..0.023 rows=1 loo...
7	Index Cond: (pts IS NOT NULL)
8	Heap Fetches: 0
9	-> Nested Loop (cost=0.59..20147.08 rows=12382 width=107) (actual time=22.381..63.622 rows=1 loops=1)
10	-> Nested Loop (cost=0.45..19776.62 rows=12382 width=46) (actual time=22.373..63.613 rows=1 loops=1)
11	-> Nested Loop (cost=0.30..19468.07 rows=12382 width=33) (actual time=22.360..63.599 rows=1 loops=1)
12	-> Seq Scan on games_details gd (cost=0.00..14585.85 rows=12382 width=25) (actual time=22.312..63.526 rows=1 loops=1)
13	Filter: (pts = $1)
14	Rows Removed by Filter: 668627
15	-> Memoize (cost=0.30..0.47 rows=1 width=16) (actual time=0.041..0.041 rows=1 loops=1)
16	Cache Key: gd.game_id
17	Cache Mode: logical
18	Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
19	-> Index Scan using games_pkey on games g (cost=0.29..0.46 rows=1 width=16) (actual time=0.034..0.034 rows=1 loops=1)
20	Index Cond: (game_id = gd.game_id)
21	-> Memoize (cost=0.15..0.17 rows=1 width=17) (actual time=0.011..0.011 rows=1 loops=1)
22	Cache Key: g.home_team_id
23	Cache Mode: logical
24	Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
25	-> Index Scan using teams_pkey on teams t1 (cost=0.14..0.16 rows=1 width=17) (actual time=0.009..0.009 rows=1 loops=1)
26	Index Cond: (team_id = g.home_team_id)
27	-> Memoize (cost=0.15..0.17 rows=1 width=17) (actual time=0.006..0.006 rows=1 loops=1)
28	Cache Key: g.visitor_team_id
29	Cache Mode: logical
30	Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
31	-> Index Scan using teams_pkey on teams t2 (cost=0.14..0.16 rows=1 width=17) (actual time=0.004..0.004 rows=1 loops=1)
32	Index Cond: (team_id = g.visitor_team_id)
33	Planning Time: 0.931 ms
34	Execution Time: 63.914 ms

Total rows: 34 of 34 | Query complete 00:00:00.104

"Limit (cost=10711.26..10727.53 rows=10 width=107) (actual time=373.644..439.477 rows=1 loops=1)"
" InitPlan 1 (returns $1)"
" -> Finalize Aggregate (cost=10710.65..10710.66 rows=1 width=4) (actual time=346.107..346.179 rows=1 loops=1)"
" -> Gather (cost=10710.44..10710.65 rows=2 width=4) (actual time=199.120..346.170 rows=3 loops=1)"
" Workers Planned: 2"
" Workers Launched: 2"
" -> Partial Aggregate (cost=9710.44..9710.45 rows=1 width=4) (actual time=87.354..87.355 rows=1 loops=3)"
" -> Parallel Seq Scan on games_details (cost=0.00..9013.95 rows=278595 width=4) (actual time=0.007..56.307 rows=222876 loops=3)"
" -> Nested Loop (cost=0.59..20147.08 rows=12382 width=107) (actual time=373.643..439.401 rows=1 loops=1)"

```
"       ->  Nested Loop  (cost=0.45..19776.62 rows=12382 width=46) (actual
time=373.634..439.391 rows=1 loops=1)"
"             ->  Nested Loop  (cost=0.30..19468.07 rows=12382 width=33) (actual
time=373.618..439.374 rows=1 loops=1)"
"                   ->  Seq Scan on games_details gd  (cost=0.00..14585.85 rows=12382 width=25)
(actual time=373.582..439.334 rows=1 loops=1)"
"                         Filter: (pts = $1)"
"                         Rows Removed by Filter: 668627"
"                   ->  Memoize  (cost=0.30..0.47 rows=1 width=16) (actual time=0.030..0.031 rows=1
loops=1)"
"                         Cache Key: gd.game_id"
"                         Cache Mode: logical"
"                         Hits: 0  Misses: 1  Evictions: 0  Overflows: 0  Memory Usage: 1kB"
"                         ->  Index Scan using games_pkey on games g  (cost=0.29..0.46 rows=1 width=16)
(actual time=0.024..0.024 rows=1 loops=1)"
"                               Index Cond: (game_id = gd.game_id)"
"             ->  Memoize  (cost=0.15..0.17 rows=1 width=17) (actual time=0.011..0.012 rows=1
loops=1)"
"                   Cache Key: g.home_team_id"
"                   Cache Mode: logical"
"                   Hits: 0  Misses: 1  Evictions: 0  Overflows: 0  Memory Usage: 1kB"
"                   ->  Index Scan using teams_pkey on teams t1  (cost=0.14..0.16 rows=1 width=17)
(actual time=0.009..0.009 rows=1 loops=1)"
"                         Index Cond: (team_id = g.home_team_id)"
"       ->  Memoize  (cost=0.15..0.17 rows=1 width=17) (actual time=0.006..0.007 rows=1
loops=1)"
"             Cache Key: g.visitor_team_id"
"             Cache Mode: logical"
"             Hits: 0  Misses: 1  Evictions: 0  Overflows: 0  Memory Usage: 1kB"
"             ->  Index Scan using teams_pkey on teams t2  (cost=0.14..0.16 rows=1 width=17)
(actual time=0.005..0.005 rows=1 loops=1)"
"                   Index Cond: (team_id = g.visitor_team_id)"
"Planning Time: 0.706 ms"
"Execution Time: 439.871 ms"
```

QUERY PLAN 1 AFTER INDEXING



	QUERY PLAN text
5	-> Limit (cost=0.42..0.45 rows=1 width=4) (actual time=0.023..0.024 rows=1 loops=1)
6	-> Index Only Scan using idx_gamesdetails_pts_desc on games_details (cost=0.42..13981.41 rows=668628 width=4) (actual time=0.023..0.023 rows=1 loo...
7	Index Cond: (pts IS NOT NULL)
8	Heap Fetches: 0
9	-> Nested Loop (cost=0.59..20147.08 rows=12382 width=107) (actual time=22.381..63.622 rows=1 loops=1)
10	-> Nested Loop (cost=0.45..19776.62 rows=12382 width=46) (actual time=22.373..63.613 rows=1 loops=1)
11	-> Nested Loop (cost=0.30..19468.07 rows=12382 width=33) (actual time=22.360..63.599 rows=1 loops=1)
12	-> Seq Scan on games_details gd (cost=0.00..14585.85 rows=12382 width=25) (actual time=22.312..63.526 rows=1 loops=1)
13	Filter: (pts = $1)
14	Rows Removed by Filter: 668627
15	-> Memoize (cost=0.30..0.47 rows=1 width=16) (actual time=0.041..0.041 rows=1 loops=1)
16	Cache Key: gd.game_id
17	Cache Mode: logical
18	Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
19	-> Index Scan using games_pkey on games g (cost=0.29..0.46 rows=1 width=16) (actual time=0.034..0.034 rows=1 loops=1)
20	Index Cond: (game_id = gd.game_id)
21	-> Memoize (cost=0.15..0.17 rows=1 width=17) (actual time=0.011..0.011 rows=1 loops=1)
22	Cache Key: g.home_team_id
23	Cache Mode: logical
24	Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
25	-> Index Scan using teams_pkey on teams t1 (cost=0.14..0.16 rows=1 width=17) (actual time=0.009..0.009 rows=1 loops=1)
26	Index Cond: (team_id = g.home_team_id)
27	-> Memoize (cost=0.15..0.17 rows=1 width=17) (actual time=0.006..0.006 rows=1 loops=1)
28	Cache Key: g.visitor_team_id
29	Cache Mode: logical
30	Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB
31	-> Index Scan using teams_pkey on teams t2 (cost=0.14..0.16 rows=1 width=17) (actual time=0.004..0.004 rows=1 loops=1)
32	Index Cond: (team_id = g.visitor_team_id)
33	Planning Time: 0.931 ms
34	Execution Time: 63.914 ms

Total rows: 34 of 34 Query complete 00:00:00.104

"Limit (cost=1.05..17.32 rows=10 width=107) (actual time=22.383..63.626 rows=1 loops=1)" "

InitPlan 2 (returns $1)"

" -> Result (cost=0.45..0.46 rows=1 width=4) (actual time=0.025..0.026 rows=1 loops=1)" "

InitPlan 1 (returns $0)"

" -> Limit (cost=0.42..0.45 rows=1 width=4) (actual time=0.023..0.024 rows=1 loops=1)"

" -> Index Only Scan using idx_gamesdetails_pts_desc on games_details
(cost=0.42..13981.41 rows=668628 width=4) (actual time=0.023..0.023 rows=1 loops=1)"
" Index Cond: (pts IS NOT NULL)"

" Heap Fetches: 0"

" -> Nested Loop (cost=0.59..20147.08 rows=12382 width=107) (actual time=22.381..63.622 rows=1 loops=1)"

" -> Nested Loop (cost=0.45..19776.62 rows=12382 width=46) (actual time=22.373..63.613 rows=1 loops=1)"

" -> Nested Loop (cost=0.30..19468.07 rows=12382 width=33) (actual time=22.360..63.599 rows=1 loops=1)"

" -> Seq Scan on games_details gd (cost=0.00..14585.85 rows=12382 width=25) (actual time=22.312..63.526 rows=1 loops=1)"

" Filter: (pts = $1)"

" Rows Removed by Filter: 668627"

" -> Memoize (cost=0.30..0.47 rows=1 width=16) (actual time=0.041..0.041 rows=1 loops=1)"

" Cache Key: gd.game_id"

" Cache Mode: logical"

" Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB"

" -> Index Scan using games_pkey on games g (cost=0.29..0.46 rows=1 width=16) (actual time=0.034..0.034 rows=1 loops=1)"

" Index Cond: (game_id = gd.game_id)"

" -> Memoize (cost=0.15..0.17 rows=1 width=17) (actual time=0.011..0.011 rows=1 loops=1)"

" Cache Key: g.home_team_id"

" Cache Mode: logical"

" Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB"

" -> Index Scan using teams_pkey on teams t1 (cost=0.14..0.16 rows=1 width=17) (actual time=0.009..0.009 rows=1 loops=1)"

" Index Cond: (team_id = g.home_team_id)"
" -> Memoize (cost=0.15..0.17 rows=1 width=17) (actual time=0.006..0.006 rows=1 loops=1)"

" Cache Key: g.visitor_team_id"

" Cache Mode: logical"

" Hits: 0 Misses: 1 Evictions: 0 Overflows: 0 Memory Usage: 1kB"

" -> Index Scan using teams_pkey on teams t2 (cost=0.14..0.16 rows=1 width=17) (actual time=0.004..0.004 rows=1 loops=1)"

" Index Cond: (team_id = g.visitor_team_id)"

"Planning Time: 0.931 ms"

"Execution Time: 63.914 ms"

PostgreSQL QUERIES

Below SQL queries are used to explore and analyze the trends and patterns of the NBS dataset

QUERIES RUN ON PostgreSQL version >= 9.6

STEPS TO REPRODUCE THE QUERIED RESULTS

1. **Access the PostgreSQL platform. Make sure the version is >= 9.6**
2. **Run the DDL files provided in the DDL_STATEMENTS zip file.**
3. **Run the DMN files provided in the DML_STATEMENTS zip file.**
4. **Make sure the dataset (NBA_DATASET) is downloaded to your local system.**
5. **Run the queries provided in the QUERIES zip file to explore the data and find the results.**
6. **The data analysis part of the queries has been added in the project report.**

1. Top 10 players with the most points in a single game and the corresponding date and opponent

```
SELECT GD.PLAYER_NAME
, GD.PTS
```

```
, G.GAME_DATE_EST
, CASE WHEN G.HOME_TEAM_ID = GD.TEAM_ID THEN T1.CITY ELSE T2.CITY END AS
OPPONENT_CITY
, CASE WHEN G.HOME_TEAM_ID = GD.TEAM_ID THEN T1.ABBREVIATION ELSE
T2.ABBREVIATION END AS OPPONENT_ABBREVIATION
FROM GAMES_DETAILS GD
JOIN GAMES G
ON GD.GAME_ID = G.GAME_ID
JOIN TEAMS T1
ON G.HOME_TEAM_ID = T1.TEAM_ID
JOIN TEAMS T2
ON G.VISITOR_TEAM_ID = T2.TEAM_ID
GROUP BY GD.PLAYER_NAME, GD.PTS, G.GAME_DATE_EST, G.HOME_TEAM_ID, GD.TEAM_ID,
T1.CITY, T2.CITY, T1.ABBREVIATION, T2.ABBREVIATION
HAVING GD.PTS > (SELECT MIN(PTS) FROM GAMES_DETAILS)
ORDER BY GD.PTS DESC
LIMIT 10;
```

2. Teams with the most wins in the last 10 seasons, along with their win percentage

```
SELECT TEAM
, WIN_PERCENT
, COUNT (*) AS NUM_SEASONS
FROM ranking
WHERE SEASON_ID >=
(SELECT MAX(SEASON_ID) - 9 FROM ranking)
GROUP BY TEAM, WIN_PERCENT
HAVING COUNT (*) >= 5
ORDER BY WIN_PERCENT DESC, NUM_SEASONS DESC
LIMIT 10;
```

3. Players who have played for multiple teams in a single season

```
SELECT DISTINCT(P_LIST.PLAYER_NAME), P_LIST.TEAM_ABBREVIATION, P_LIST.GAME_ID
FROM
(SELECT PLAYER_NAME
 , TEAM_ABBREVIATION
```

```
 , GAME_ID
 , COUNT(TEAM_ID) OVER (PARTITION BY GAME_ID, PLAYER_ID) AS NUM_TEAMS
 FROM GAMES_DETAILS
) P_LIST
 WHERE NUM_TEAMS > 1
 ORDER BY P_LIST.PLAYER_NAME;
```

4. Teams with the highest average points per game at home and away

```
SELECT T.CITY
, ROUND (AVG (CASE WHEN G.HOME_TEAM_ID = T.TEAM_ID THEN GD.PTS ELSE 0 END),2) AS
AVG_HOME_POINTS
, ROUND (AVG (CASE WHEN G.VISITOR_TEAM_ID = T.TEAM_ID THEN GD.PTS ELSE 0 END),2) AS
AVG_AWAY_POINTS
FROM GAMES_DETAILS GD
JOIN GAMES G ON GD.GAME_ID = G.GAME_ID
JOIN TEAMS T ON GD.TEAM_ID = T.TEAM_ID
GROUP BY T.CITY
ORDER BY avg_home_points DESC, avg_away_points DESC;
```

5. Teams that have won at least 60% of their games in every season in which they have participated

```
SELECT team_id, team, COUNT (WIN_PERCENT) AS MIN_WIN_PCT
FROM RANKING
WHERE WIN_PERCENT >= 0.6
GROUP BY TEAM_ID, TEAM
```

6. Top 20 players with the highest points in the NBA based on the given dataset? Who is the GOAT?

```
SELECT PLAYER_NAME, SUM(PTS) AS TOTAL_POINTS
FROM GAMES_DETAILS
GROUP BY PLAYER_NAME
```

```
ORDER BY TOTAL_POINTS DESC
LIMIT 20;
```

7. Retrieve the players with the lowest points per game average for a single season.

```
SELECT
   p.PLAYER_NAME,
   ROUND (AVG (gd.PTS),2) AS AVG_PTS_PER_GAME,
COUNT (DISTINCT g.GAME_ID) AS GAMES_PLAYED,
g.SEASON
FROM
   Player p
   JOIN Games_Details gd ON p.PLAYER_ID = gd.PLAYER_ID
   JOIN Games g ON gd.GAME_ID = g.GAME_ID
GROUP BY
   p.PLAYER_NAME, g.SEASON
HAVING
   COUNT (DISTINCT g.GAME_ID) >= 20
ORDER BY
   AVG_PTS_PER_GAME ASC
LIMIT 10;
```

8. Retrieve the players who have averaged at least 20 points per game in every season they have played (minimum of 2 seasons)

```
SELECT p.PLAYER_NAME
FROM Player p
WHERE p.PLAYER_ID IN (
   SELECT gd.PLAYER_ID
   FROM GAMES_DETAILS gd
   INNER JOIN GAMES g ON gd.GAME_ID = g.GAME_ID
INNER JOIN (
      SELECT gd.PLAYER_ID, g.SEASON, AVG(gd.PTS) AS AVG_PTS
      FROM GAMES_DETAILS gd
      INNER JOIN GAMES g ON gd.GAME_ID = g.GAME_ID
      GROUP BY gd.PLAYER_ID, g.SEASON
   ) t ON gd.PLAYER_ID = t.PLAYER_ID AND g.SEASON = t.SEASON
   GROUP BY gd.PLAYER_ID
```

```
    HAVING MIN (t.AVG_PTS) >= 20 AND COUNT (DISTINCT g.SEASON) >= 2
);
```

9. Retrieve the players who have scored 50 or more points in a game at least 5 times in their career.

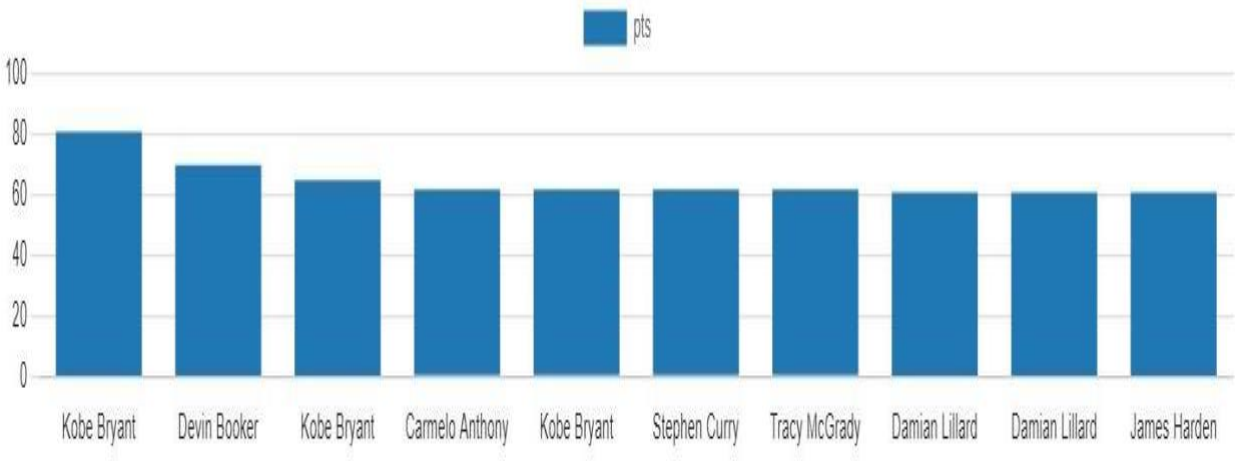
```
SELECT p.PLAYER_NAME, COUNT(*) as num_times
FROM Player p
JOIN GAMES_DETAILS gd ON p.PLAYER_ID = gd.PLAYER_ID
WHERE gd.PTS >= 50
GROUP BY p.PLAYER_NAME
HAVING COUNT (*) >= 5
ORDER BY num_times DESC
```

10. Top 5 players with the highest average points per game (PPG) in the last 3 seasons.

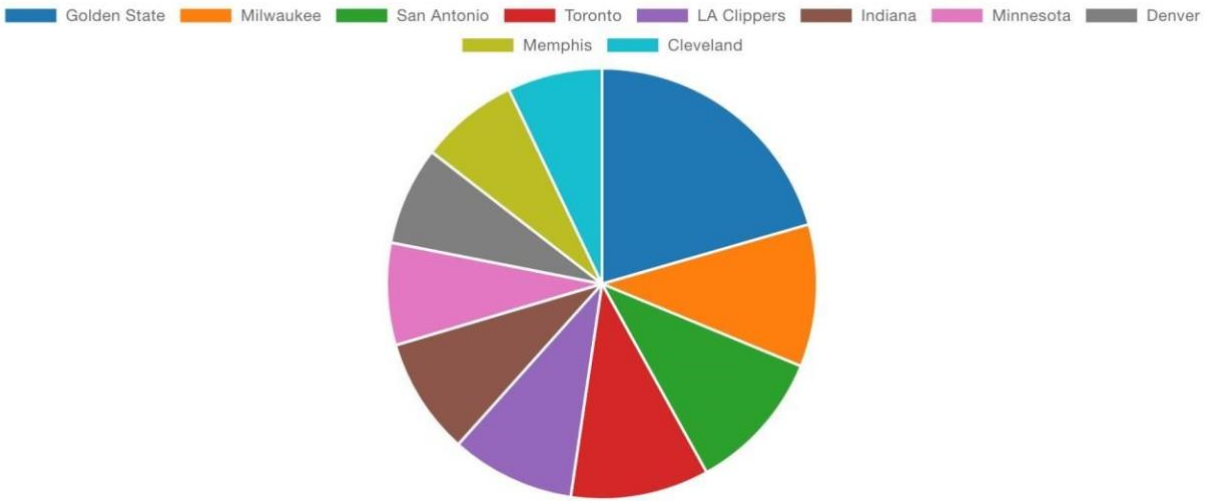
```
SELECT t.NICKNAME, ROUND(AVG(g.PTS_HOME),2) AS AVG_PTS_HOME
FROM Games g
JOIN Teams t ON g.HOME_TEAM_ID = t.TEAM_ID
WHERE g.GAME_DATE_EST >= '2020-01-01' AND g.GAME_DATE_EST < '2023-01-01'
GROUP BY t.NICKNAME
ORDER BY AVG_PTS_HOME DESC
LIMIT 5;
```

SUMMARY

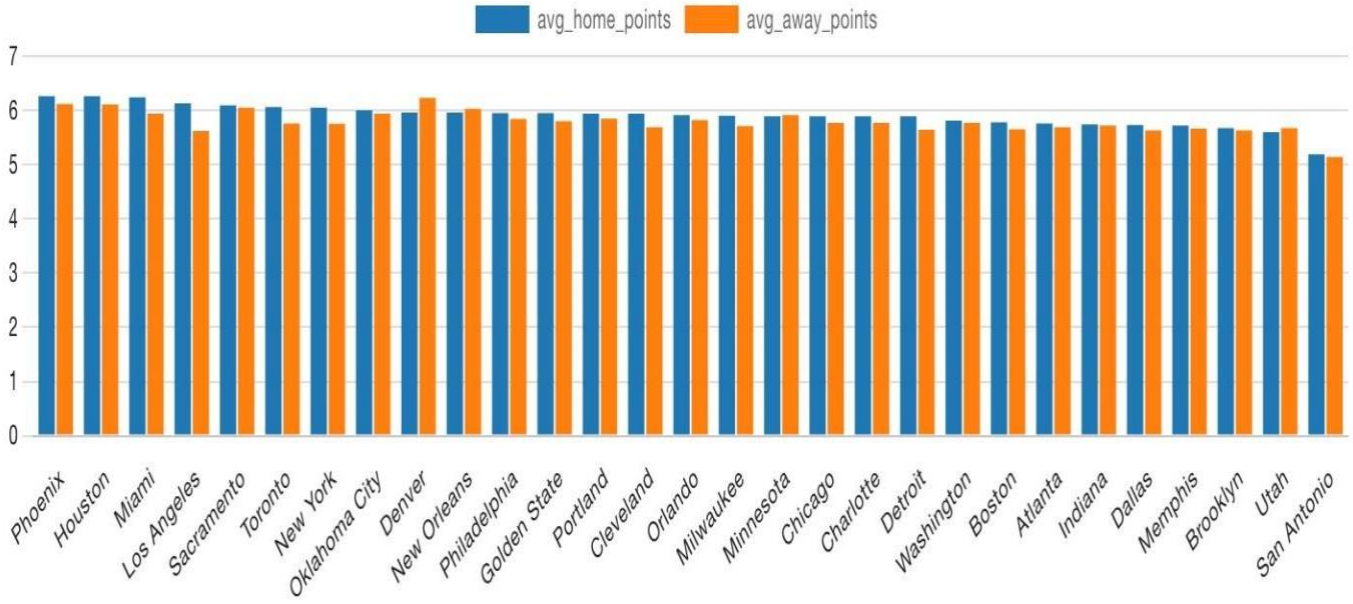
Query 1: Top 10 players with the most points in a single game



Query 2: Teams with the most wins in the last 10 seasons, along with their win percentage

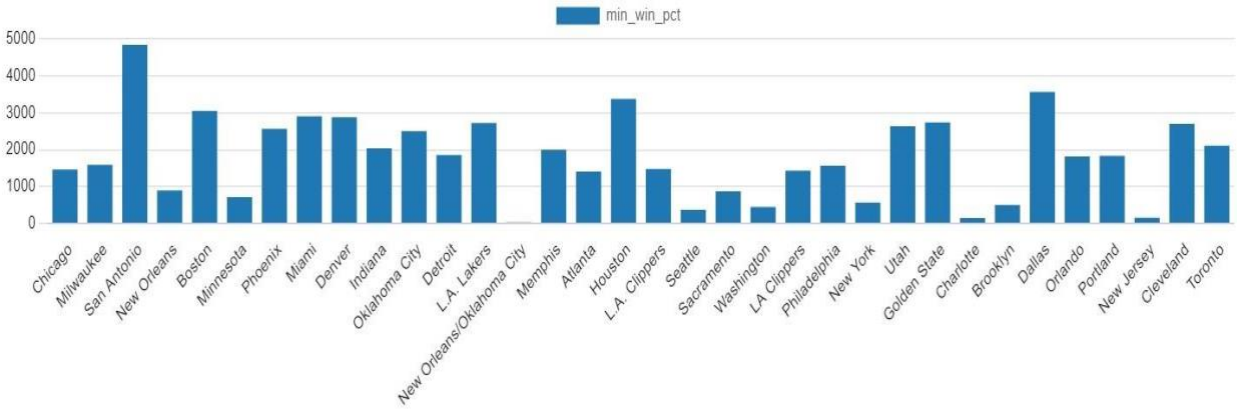


Query 4: Teams with the highest average points per game at home and away



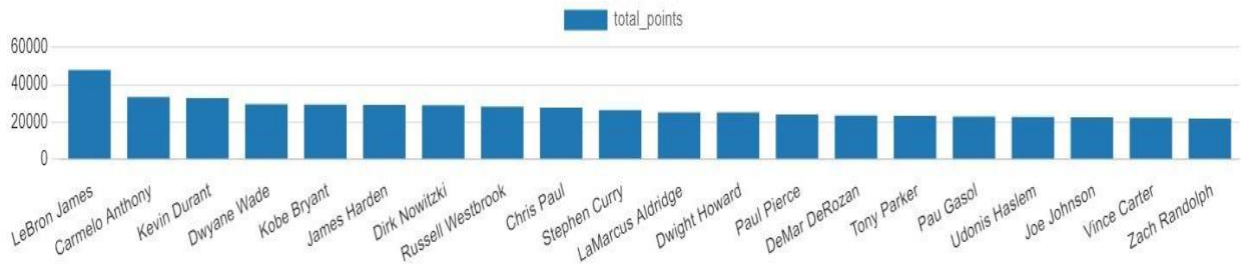
Query 5:

Teams that have won at least 60% of their games in total in which they have participated:

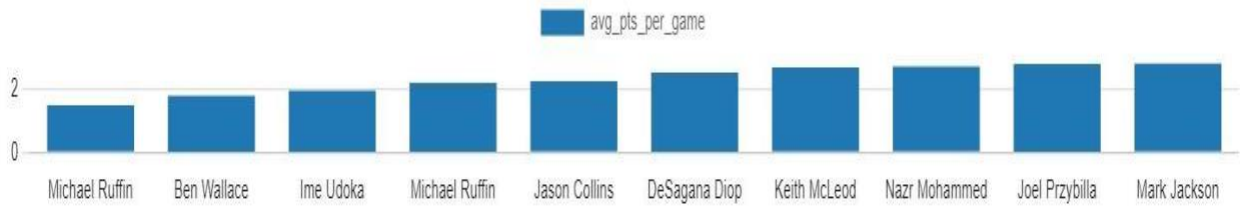


Query 6:

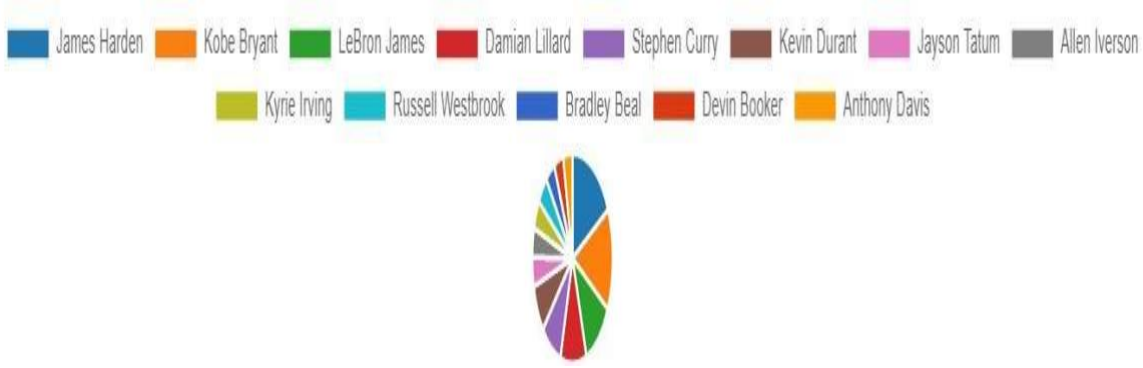
Top 20 players with the highest points in the NBA based on the dataset available? Who is the GOAT?



Query 7: Players with the lowest points per game average for a single season



Query 9: Players who have scored 50 or more points in a game at least 5 times in their career



Query 10: Top 5 players with the highest average points per game (PPG) in the last 3 seasons.

SELECT Player.PLAYER_NAME, Teams.NICKNAME, AVG(Games_Details.PTS) AS AVG_PPG

```
FROM Player
JOIN Games_Details ON Player.PLAYER_ID = Games_Details.PLAYER_ID
JOIN Teams ON Games_Details.TEAM_ID = Teams.TEAM_ID
JOIN GAMES ON GAMES.GAME_ID = Games_Details.GAME_ID
WHERE GAMES.SEASON >= 2019
GROUP BY Player.PLAYER_NAME, Teams.NICKNAME
ORDER BY AVG_PPG DESC
LIMIT 5;
```

FINAL CONCLUSIONS

Ultimately, this project was a success!!

Exploratory data analysis on the NBA dataset was procured successfully and a statistical and visual representation of the performance of different teams, players, game details, and ranking based on the points and performance in the NBA game from the year/season 2014 till the year 2022 was also provided.

Based on the analysis, it can be seen that the queries covered a variety of topics, including retrieving information on top-scoring players, teams with the highest win percentage, games that went into overtime, and more.

The schema used in these examples, consisting of five tables, is a simplified version of a database that could be used to store NBA game data. It includes tables for teams, games, players, and rankings, with additional information stored in the Games_details table.

Teams and players which have the worst performance have to find the steps to improve their performance. Overall, these queries demonstrated the flexibility and power of SQL in extracting meaningful insights from large and complex datasets. By using PostgreSQL, we can quickly and easily retrieve the information we need to make informed decisions and gain a deeper understanding of the data.