5/29/2023

Database Project

Liav Ariel -212830871-
Amiad Korman -212608442-

Table of Contents

Description of the Organization

During the initial phase of setting up a basketball league, various essential data needs to be stored in a database to ensure efficient management and organization. The following data points are crucial:

1. League Information: Details about the league itself, such as its name, start and end dates, location, and any specific rules or regulations.

   Team Information: Each participating team's data should be stored, including their unique identifier, team name, logo, home arena, coach, and roster of players.

2. Player Information: For each player, their personal details such as name, age, height, weight, position, nationality, and contact information should be recorded. It may also be necessary to store additional data, like player headshots or medical records.

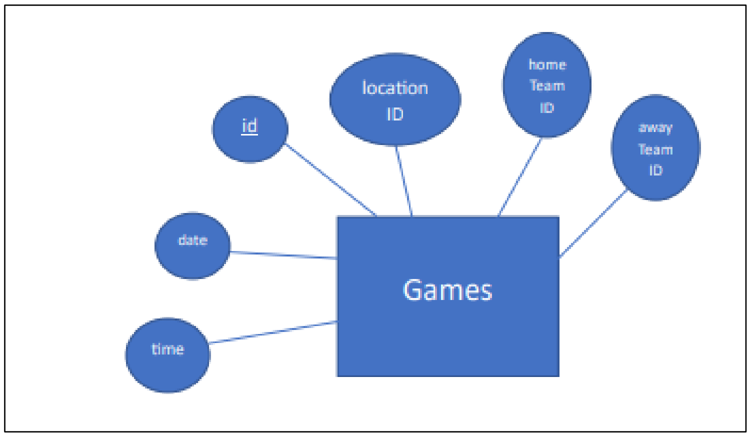
Once the league is established and teams are registered, the focus shifts to managing the games and tracking player and team statistics. The database should include the following data:

1. Game Schedule: A comprehensive schedule with game dates, start times, and locations. This enables easy access to information about upcoming games and past results.
2. Game Results: For each game, the final score, team statistics (such as field goal percentage, rebounds, assists, etc.), and individual player statistics (points scored, assists, rebounds, steals, blocks, etc.) should be recorded. This data helps in analyzing team and player performance.
3. Awards and Recognitions: Information regarding awards such as MVP (Most Valuable Player), Rookie of the Year, Defensive Player of the Year, and other accolades should be stored. This includes the recipient's name, team affiliation, and the season in which the award was earned.
4. Statistical Records: Maintaining historical statistical records can be valuable for comparison and analysis. This involves storing all-time records, such as the most points scored in a game, highest scoring average, career assists leaders, and other significant achievements.

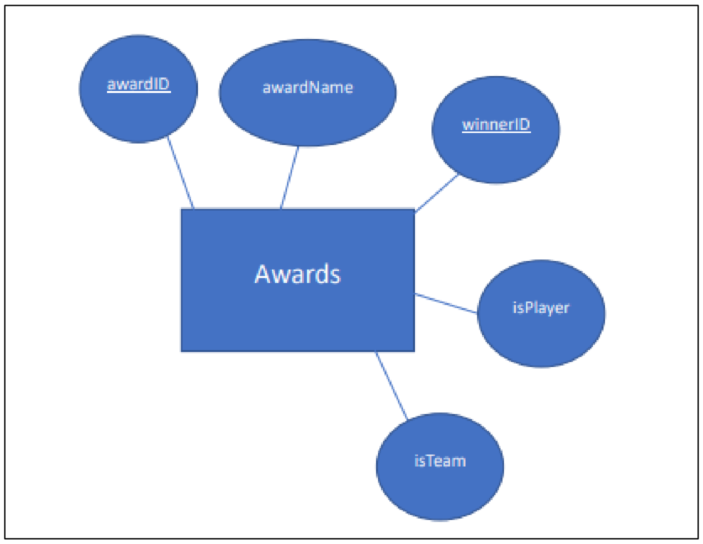
By collecting and storing these data points in a well-structured database, the basketball league can effectively manage team registrations, track game results and statistics, evaluate player and team performance, and maintain a historical record of achievements.

In our project, we will focus on the tables: Games, Game Team Stats, Awards.

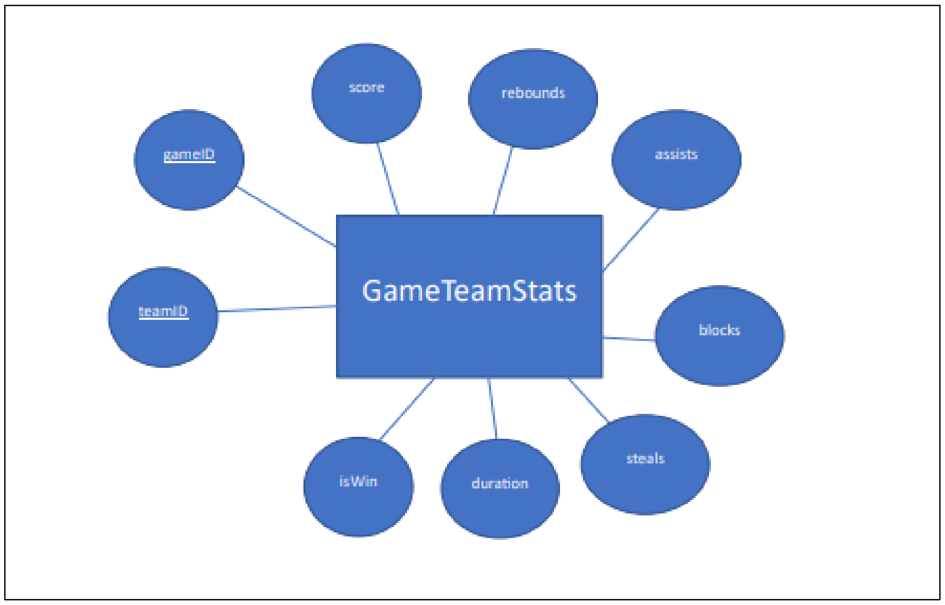
ERD Chart



id
location ID
home Team ID
away Team ID
date
Games
time



awardID
awardName
winnerID
Awards
isPlayer
isTeam



score
rebounds
gameID
assists
GameTeamStats
teamID
blocks
steals
isWin
duration

Description of the Entities

Games

The "Games" entity represents basketball games within the league. It serves to store information about each game, including the following attributes:

gameID (INT, PRIMARY KEY)	This attribute represents the unique identifier for each game. It is typically an auto-incremented numerical value assigned to each game entry in the database
locationID (INT, NOT NULL)	This attribute refers to the unique identifier of the location or venue where the game is being held. It may be linked to a separate "Locations" entity that stores information about each venue, such as its name, address, capacity, etc.
HomeTeamID (INT, NOT NULL)	This attribute represents the unique identifier of the home team participating in the game. It can be linked to the "Teams" entity, which stores information about all the teams in the league, including their team name, coach, and roster.
AwayTeamID (INT, NOT NULL)	This attribute represents the unique identifier of the home team participating in the game. It can be linked to the "Teams" entity, which stores information about all the teams in the league, including their team name, coach, and roster.
GameDate (Date, NOT NULL)	This attribute represents the date on which the game is scheduled to take place. It is typically stored in a date format that allows easy sorting and comparison.

GameTeamStats

The "GameTeamStats" table represents the statistical data associated with each team's performance in a specific basketball game. It stores information about various statistics and attributes related to the team's performance. Here is a breakdown of the attributes within the "GameTeamStats" table:

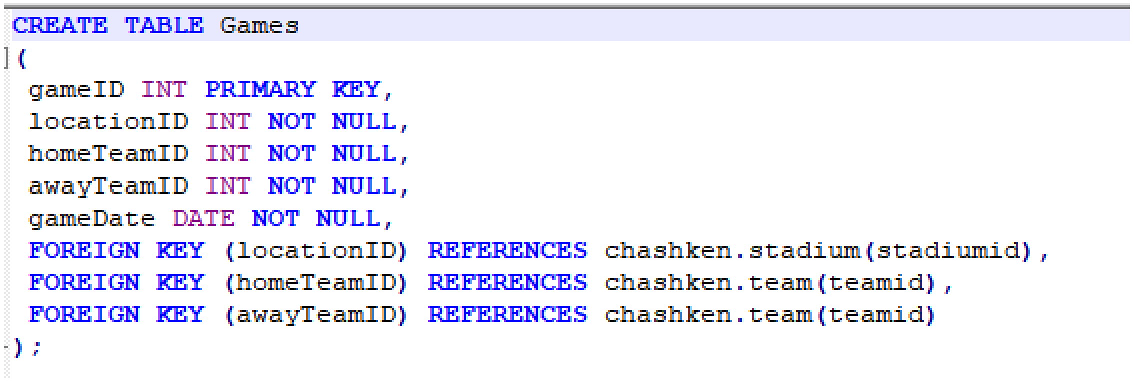
GameTeamStatsID**(INT, PRIMARY KEY)**	
gameID **(INT, NOT NULL)**	This attribute refers to the unique identifier of the game to which the statistics belong. It is linked to the "Games" entity, enabling the association of game-specific information, such as location, date, and time.
teamID **(INT, NOT NULL)**	This attribute represents the unique identifier of the team. It is linked to the "Teams" entity, allowing easy retrieval of team information such as team name, coach, and roster.
score **(INT, NOT NULL)**	This attribute represents the total score achieved by the team in the game.
rebounds **(INT, NOT NULL)**	This attribute represents the total number of rebounds secured by the team.
assists **(INT, NOT NULL)**	This attribute denotes the total number of assists recorded by the team.
blocks **(INT, NOT NULL)**	This attribute represents the total number of blocks achieved by the team
steals **(INT, NOT NULL)**	This attribute represents the total number of steals made by the team.
duration **(INT, NOT NULL)**	This attribute denotes the duration of the game. It can be stored as a time value to track the length of the game, typically in minutes.
isWin **(INT, NOT NULL)**	This attribute is a binary indicator (e.g., 0 or 1) that represents whether the team won the game. A value of 1 indicates a win, while a value of 0 indicates a loss.

Awards

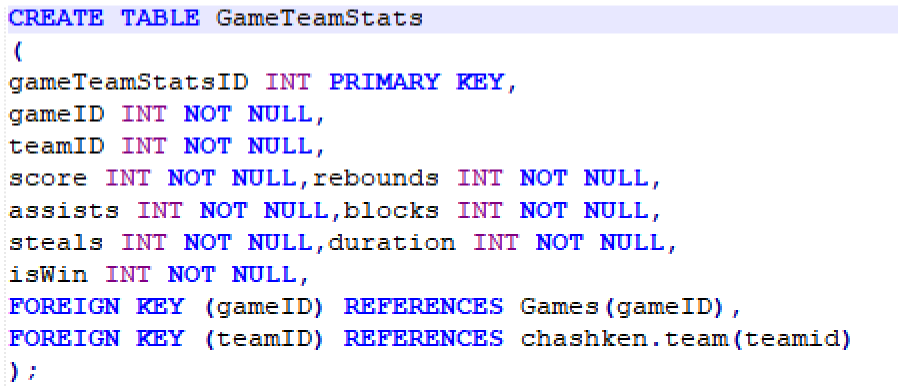
The "Awards" table represents the awards given within the basketball league to recognize outstanding achievements. It stores information about different awards and their recipients. Here is a breakdown of the attributes within the "Awards" table:

awardID **(INT, PRIMARY KEY)**	This attribute represents the unique identifier for each award. It is typically an auto-incremented numerical value assigned to each award entry in the database.
winnerID **(INT, NOT NULL)**	This attribute refers to the unique identifier of the award winner. The winnerID can be linked to either a "Players" entity or a "Teams" entity, depending on whether the award is given to an individual player or a team.
awardName **(VARCHAR(30), NOT NULL)**	This attribute stores the name or title of the award. It provides a descriptive label for the specific recognition or honor bestowed upon the winner.
isPlayer **(INT, NOT NULL)**	This attribute is a boolean indicator (e.g., true or false) that signifies whether the award is given to an individual player. A value of true indicates that the award is for a player, while a value of false indicates that the award is for a team.
isTeam **(INT, NOT NULL)**	This attribute is a boolean indicator that signifies whether the award is given to a team. A value of true indicates that the award is for a team, while a value of false indicates that the award is for an individual player.

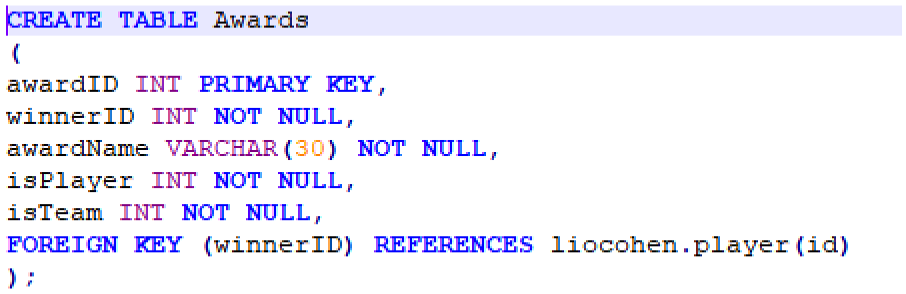
Scripts to create tables



```
CREATE TABLE Games
(
 gameID INT PRIMARY KEY,
 locationID INT NOT NULL,
 homeTeamID INT NOT NULL,
 awayTeamID INT NOT NULL,
 gameDate DATE NOT NULL,
 FOREIGN KEY (locationID) REFERENCES chashken.stadium(stadiumid),
 FOREIGN KEY (homeTeamID) REFERENCES chashken.team(teamid),
 FOREIGN KEY (awayTeamID) REFERENCES chashken.team(teamid)
);
```

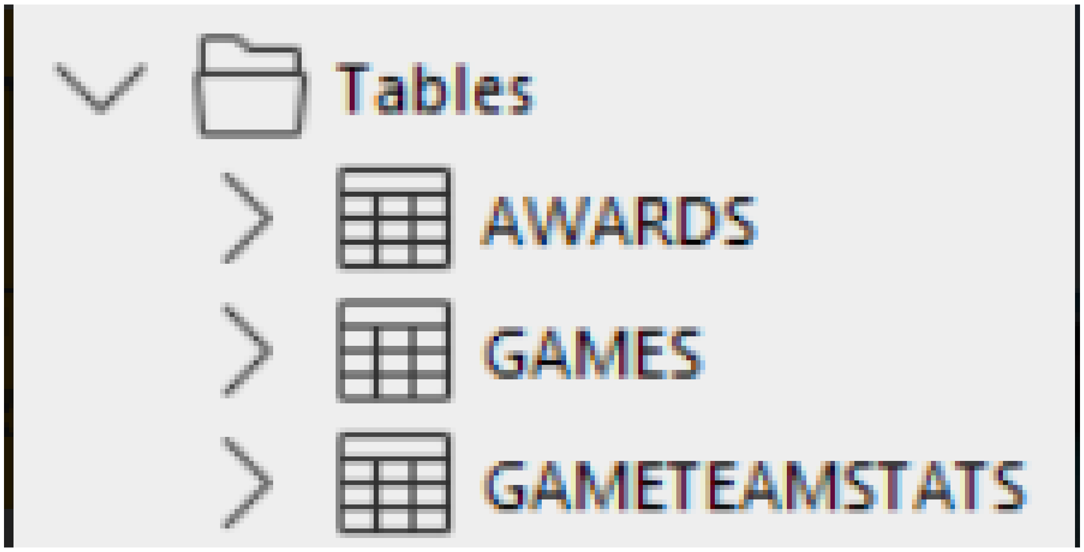


```
CREATE TABLE GameTeamStats
(
gameTeamStatsID INT PRIMARY KEY,
gameID INT NOT NULL,
teamID INT NOT NULL,
score INT NOT NULL,rebounds INT NOT NULL,
assists INT NOT NULL,blocks INT NOT NULL,
steals INT NOT NULL,duration INT NOT NULL,
isWin INT NOT NULL,
FOREIGN KEY (gameID) REFERENCES Games(gameID),
FOREIGN KEY (teamID) REFERENCES chashken.team(teamid)
);
```

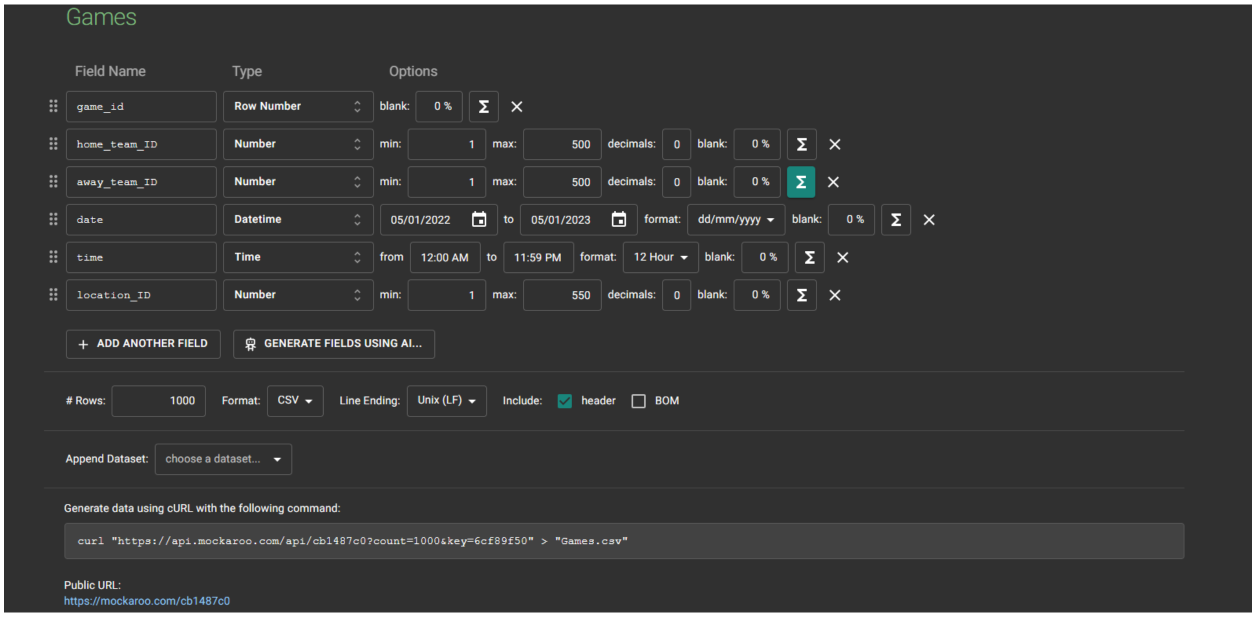


```
CREATE TABLE Awards
(
awardID INT PRIMARY KEY,
winnerID INT NOT NULL,
awardName VARCHAR(30) NOT NULL,
isPlayer INT NOT NULL,
isTeam INT NOT NULL,
FOREIGN KEY (winnerID) REFERENCES liocohen.player(id)
);
```

Results


Tables
AWARDS
GAMES
GAMETEAMSTATS

Generate Data to Tables (using Mockaroo)



Games

Field Name	Type	Options
game_id	Row Number	blank: 0 %
home_team_ID	Number	min: 1 max: 500 decimals: 0 blank: 0 %
away_team_ID	Number	min: 1 max: 500 decimals: 0 blank: 0 %
date	Datetime	05/01/2022 to 05/01/2023 format: dd/mm/yyyy blank: 0 %
time	Time	from 12:00 AM to 11:59 PM format: 12 Hour blank: 0 %
location_ID	Number	min: 1 max: 550 decimals: 0 blank: 0 %

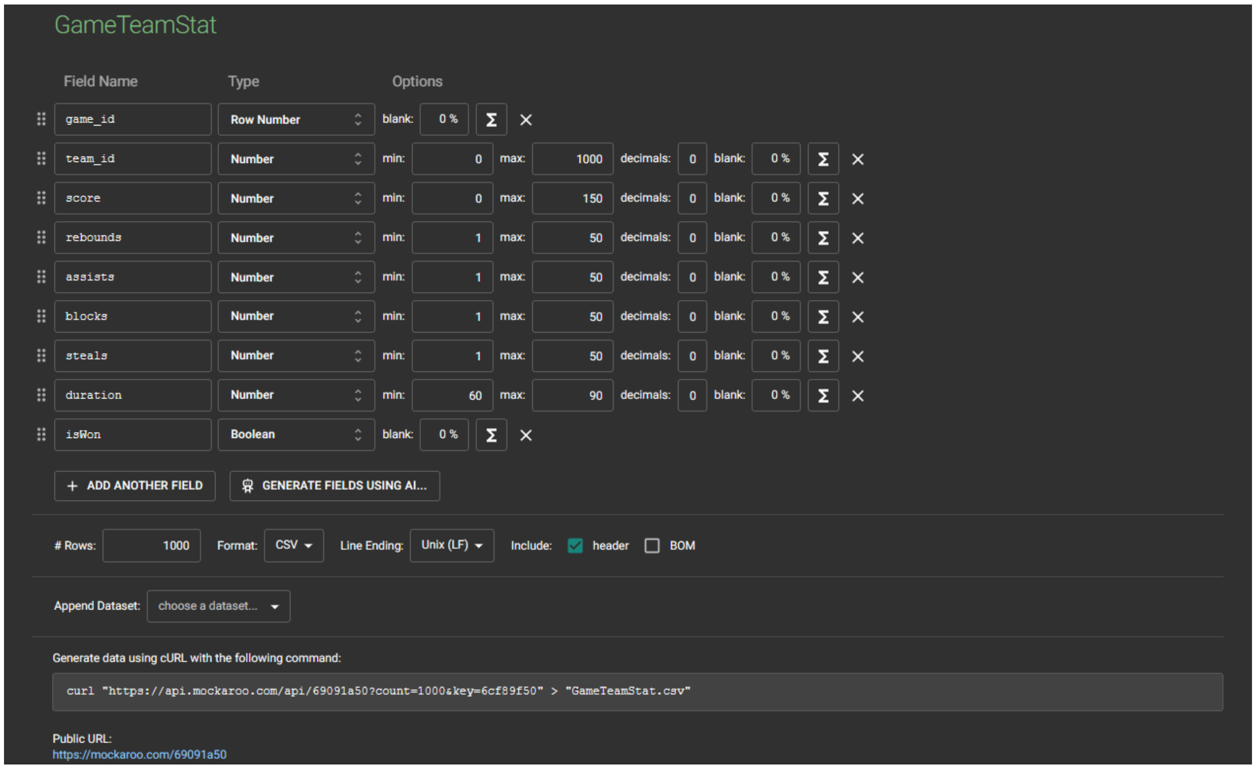
Rows: 1000 Format: CSV Line Ending: Unix (LF) Include: ✓ header ☐ BOM

Append Dataset: choose a dataset...

Generate data using cURL with the following command:

```
curl "https://api.mockaroo.com/api/cb1487c0?count=1000&key=6cf89f50" > "Games.csv"
```

Public URL:
https://mockaroo.com/cb1487c0



GameTeamStat

Field Name	Type	Options
game_id	Row Number	blank: 0 %
team_id	Number	min: 0 max: 1000 decimals: 0 blank: 0 %
score	Number	min: 0 max: 150 decimals: 0 blank: 0 %
rebounds	Number	min: 1 max: 50 decimals: 0 blank: 0 %
assists	Number	min: 1 max: 50 decimals: 0 blank: 0 %
blocks	Number	min: 1 max: 50 decimals: 0 blank: 0 %
steals	Number	min: 1 max: 50 decimals: 0 blank: 0 %
duration	Number	min: 60 max: 90 decimals: 0 blank: 0 %
isWon	Boolean	blank: 0 %

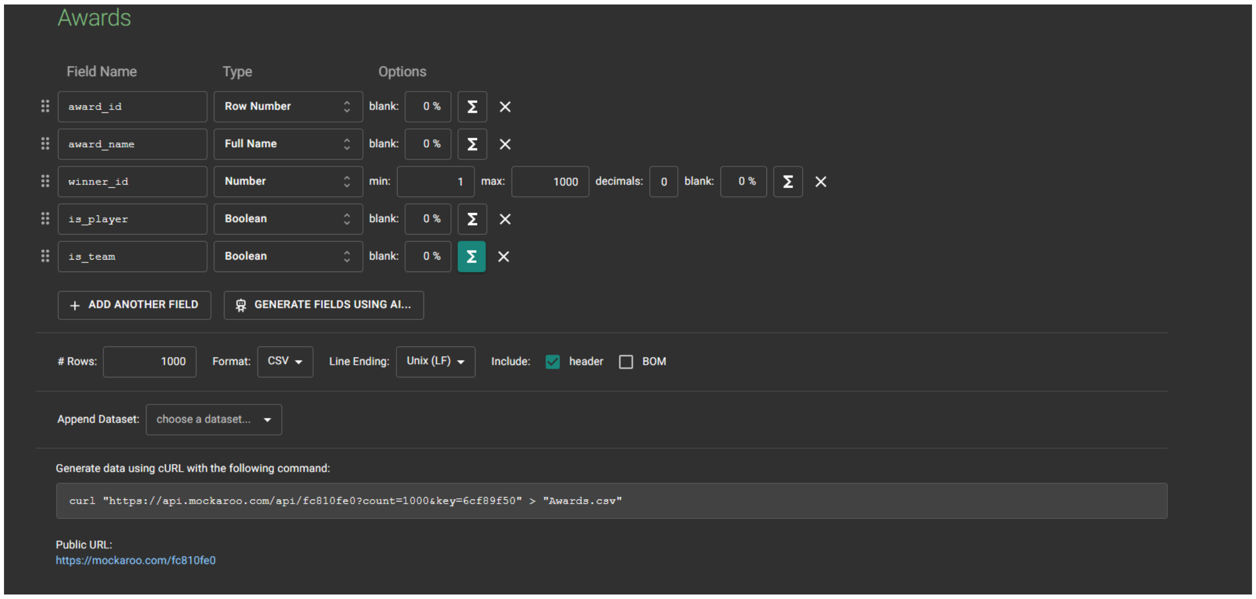
Rows: 1000 Format: CSV Line Ending: Unix (LF) Include: ✓ header ☐ BOM

Append Dataset: choose a dataset...

Generate data using cURL with the following command:

```
curl "https://api.mockaroo.com/api/69091a50?count=1000&key=6cf89f50" > "GameTeamStat.csv"
```

Public URL:
https://mockaroo.com/69091a50



Awards

Field Name	Type	Options
award_id	Row Number	blank: 0 %
award_name	Full Name	blank: 0 %
winner_id	Number	min: 1 max: 1000 decimals: 0 blank: 0 %
is_player	Boolean	blank: 0 %
is_team	Boolean	blank: 0 %

Rows: 1000 Format: CSV Line Ending: Unix (LF) Include: [✓ header] BOM

Append Dataset: choose a dataset...

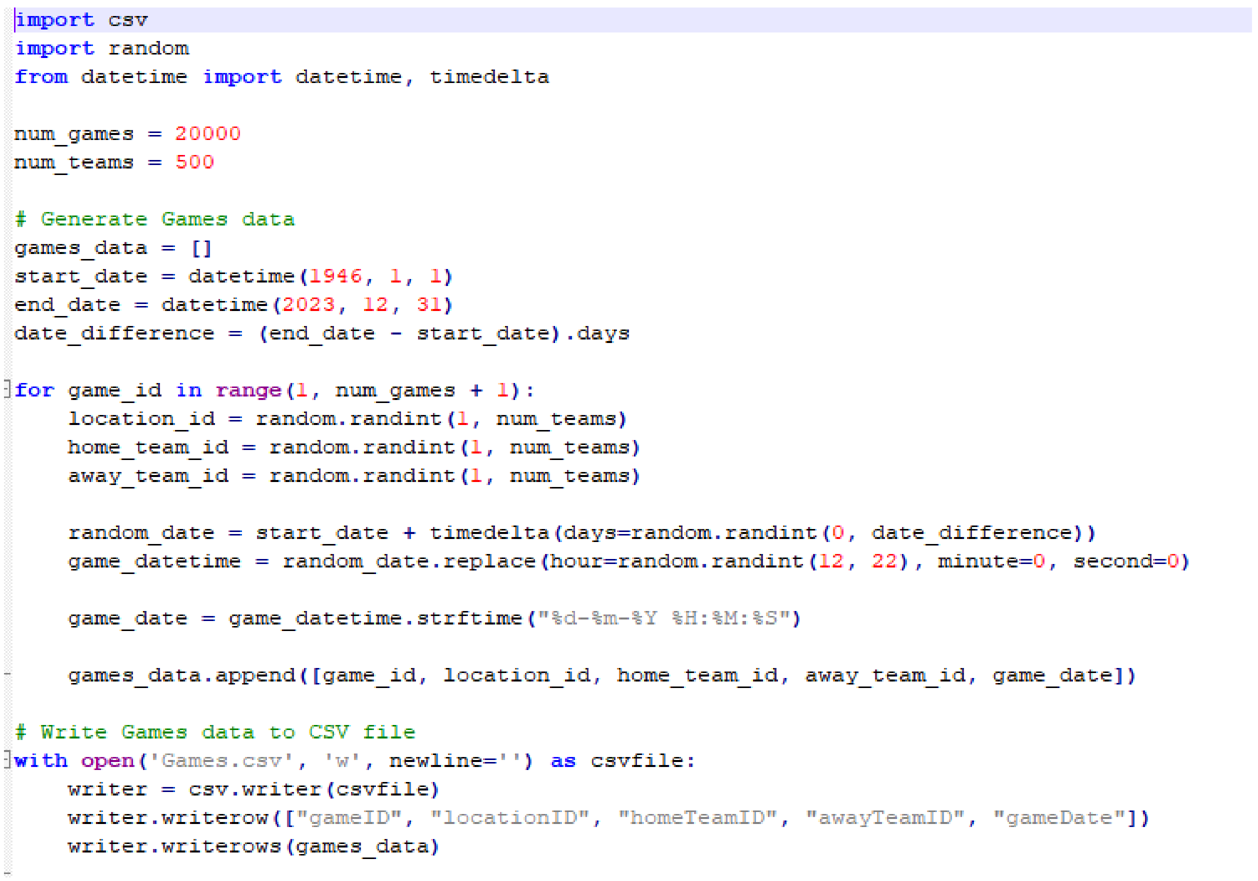
Generate data using cURL with the following command:

```
curl "https://api.mockaroo.com/api/fc810fe0?count=1000&key=6cf89f50" > "Awards.csv"
```

Public URL:
https://mockaroo.com/fc810fe0

Generate Data to Tables (using Python)

Games



```python
import csv
import random
from datetime import datetime, timedelta

num_games = 20000
num_teams = 500

# Generate Games data
games_data = []
start_date = datetime(1946, 1, 1)
end_date = datetime(2023, 12, 31)
date_difference = (end_date - start_date).days

for game_id in range(1, num_games + 1):
    location_id = random.randint(1, num_teams)
    home_team_id = random.randint(1, num_teams)
    away_team_id = random.randint(1, num_teams)

    random_date = start_date + timedelta(days=random.randint(0, date_difference))
    game_datetime = random_date.replace(hour=random.randint(12, 22), minute=0, second=0)

    game_date = game_datetime.strftime("%d-%m-%Y %H:%M:%S")

    games_data.append([game_id, location_id, home_team_id, away_team_id, game_date])

# Write Games data to CSV file
with open('Games.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["gameID", "locationID", "homeTeamID", "awayTeamID", "gameDate"])
    writer.writerows(games_data)
```

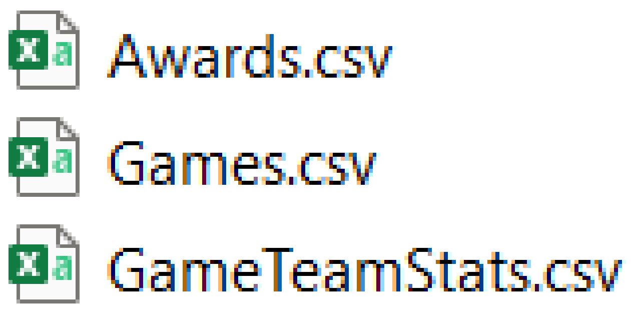
GameTeamStats

```
import csv
import random

# Read Games data from Games.csv
games_data = []
with open('Games.csv', 'r') as csvfile:
    reader = csv.reader(csvfile)
    next(reader)  # Skip header row
    for row in reader:
        games_data.append(row)

# Generate GameTeamStats data
gameteamstats_data = []
for game in games_data:
    game_id = int(game[0])
    home_team_id = int(game[2])
    away_team_id = int(game[3])
    game_home_stat_id = game_id * 2 - 1
    game_away_stat_id = game_id * 2
    # Generate random scores for each team
    home_team_score = random.randint(70, 120)
    away_team_score = random.randint(70, 120)

    # Determine the winner based on the scores
    if home_team_score > away_team_score:
        winning_team_id = home_team_id
        losing_team_id = away_team_id
        is_win = 1
    else:
        winning_team_id = away_team_id
        losing_team_id = home_team_id
        is_win = 0

    # Team 1 (home team)
    gameteamstats_data.append([game_home_stat_id, game_id, home_team_id, home_team_score, random.randint(30, 50),
                               random.randint(15, 30), random.randint(2, 8), random.randint(5, 15), 48,
                               is_win])

    # Team 2 (away team)
    gameteamstats_data.append([game_away_stat_id, game_id, away_team_id, away_team_score, random.randint(30, 50),
                               random.randint(15, 30), random.randint(2, 8), random.randint(5, 15), 48,
                               1 - is_win])

# Write GameTeamStats data to CSV file
with open('GameTeamStats.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(
        ["gameTeamStatsID", "gameID", "teamID", "score", "rebounds", "assists", "blocks", "steals", "duration", "isWin"])
    writer.writerows(gameteamstats_data)
```

Awards

```python
import csv
import random

num_awards = 1000
num_players = 20000
num_teams = 500

# Generate Awards data
awards_data = []
for award_id in range(1, num_awards + 1):
    winner_id = random.randint(1, num_players) # There is a lot more players than teams
    if winner_id <= num_teams:
        is_player = random.randint(0, 1)
        is_team = 1 - is_player
    else:
        is_player = 1
        is_team = 0
    award_name = f"Award {award_id}"

    awards_data.append([award_id, winner_id, award_name, is_player, is_team])

# Write Awards data to CSV file
with open('Awards.csv', 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["awardID", "winnerID", "awardName", "isPlayer", "isTeam"])
    writer.writerows(awards_data)
```

Results



Awards.csv

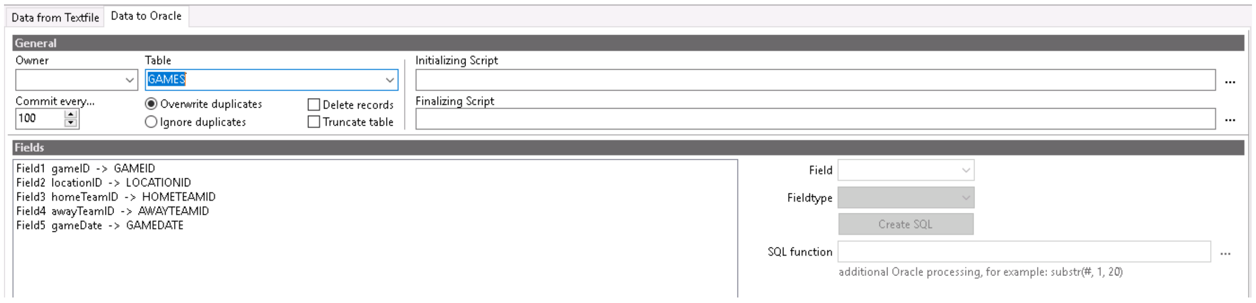
Games.csv

GameTeamStats.csv

Inserting Data Using Text-Importer

Inserting data in the Games table, using the Text-Importer of PLSQL.

The configuration of the data:



Data from Textfile | Data to Oracle

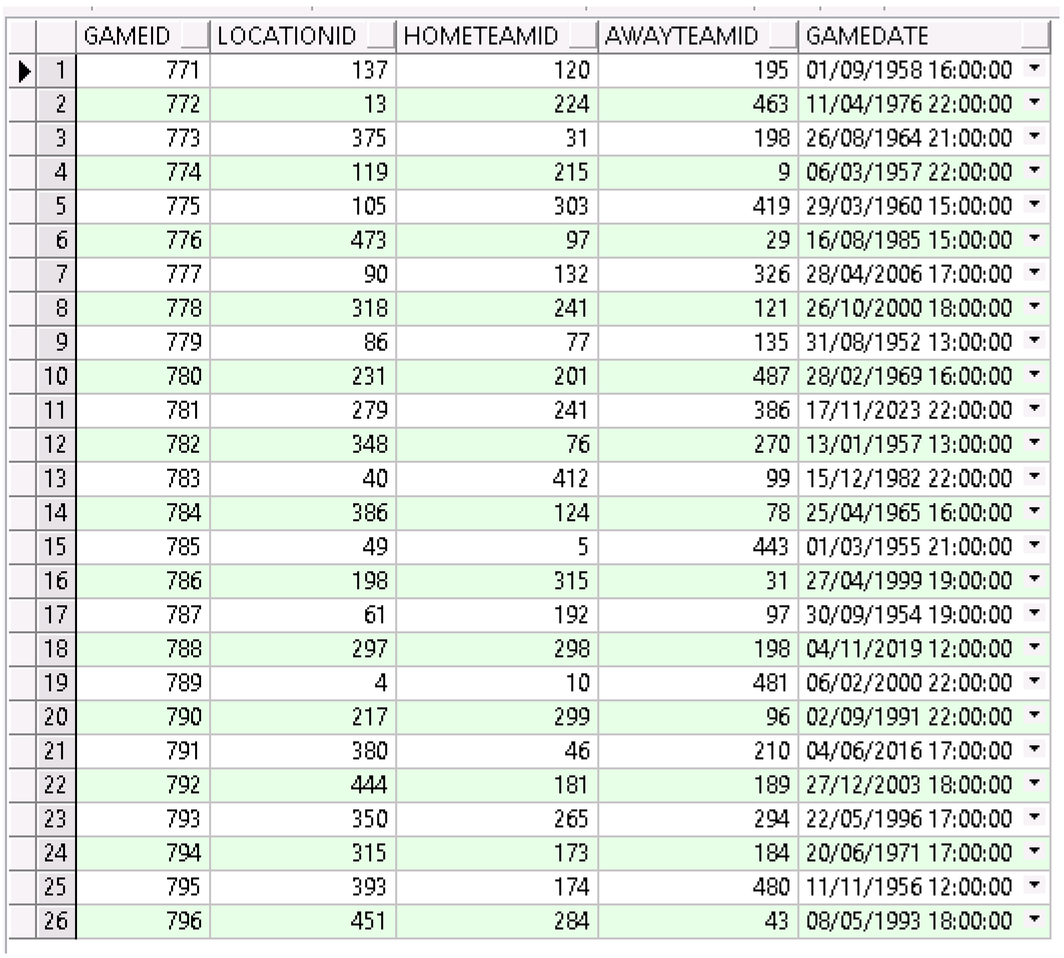
General

Owner:
Table: GAMES
Commit every... 100
Overwrite duplicates [✓] / Ignore duplicates
☐ Delete records
☐ Truncate table
Initializing Script
Finalizing Script

Fields

```
Field1 gameID -> GAMEID
Field2 locationID -> LOCATIONID
Field3 homeTeamID -> HOMETEAMID
Field4 awayTeamID -> AWAYTEAMID
Field5 gameDate -> GAMEDATE
```

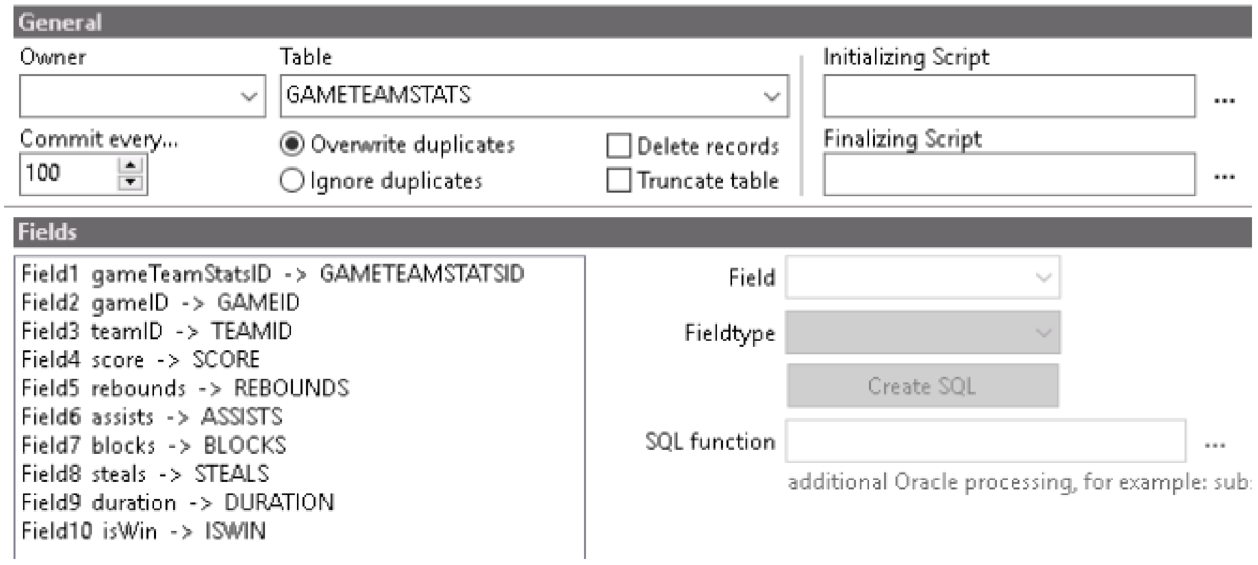
Result:



	GAMEID	LOCATIONID	HOMETEAMID	AWAYTEAMID	GAMEDATE
1	771	137	120	195	01/09/1958 16:00:00
2	772	13	224	463	11/04/1976 22:00:00
3	773	375	31	198	26/08/1964 21:00:00
4	774	119	215	9	06/03/1957 22:00:00
5	775	105	303	419	29/03/1960 15:00:00
6	776	473	97	29	16/08/1985 15:00:00
7	777	90	132	326	28/04/2006 17:00:00
8	778	318	241	121	26/10/2000 18:00:00
9	779	86	77	135	31/08/1952 13:00:00
10	780	231	201	487	28/02/1969 16:00:00
11	781	279	241	386	17/11/2023 22:00:00
12	782	348	76	270	13/01/1957 13:00:00
13	783	40	412	99	15/12/1982 22:00:00
14	784	386	124	78	25/04/1965 16:00:00
15	785	49	5	443	01/03/1955 21:00:00
16	786	198	315	31	27/04/1999 19:00:00
17	787	61	192	97	30/09/1954 19:00:00
18	788	297	298	198	04/11/2019 12:00:00
19	789	4	10	481	06/02/2000 22:00:00
20	790	217	299	96	02/09/1991 22:00:00
21	791	380	46	210	04/06/2016 17:00:00
22	792	444	181	189	27/12/2003 18:00:00
23	793	350	265	294	22/05/1996 17:00:00
24	794	315	173	184	20/06/1971 17:00:00
25	795	393	174	480	11/11/1956 12:00:00
26	796	451	284	43	08/05/1993 18:00:00

Inserting data in the GameTeamStats table, using the Text-Importer of PLSQL.

<u>The configuration of the data:</u>



General

Owner:

Table: GAMETEAMSTATS

Initializing Script:

Commit every... 100

[✓ Overwrite duplicates] Ignore duplicates

☐ Delete records

☐ Truncate table

Finalizing Script:

Fields

```
Field1 gameTeamStatsID -> GAMETEAMSTATSID
Field2 gameID -> GAMEID
Field3 teamID -> TEAMID
Field4 score -> SCORE
Field5 rebounds -> REBOUNDS
Field6 assists -> ASSISTS
Field7 blocks -> BLOCKS
Field8 steals -> STEALS
Field9 duration -> DURATION
Field10 isWin -> ISWIN
```

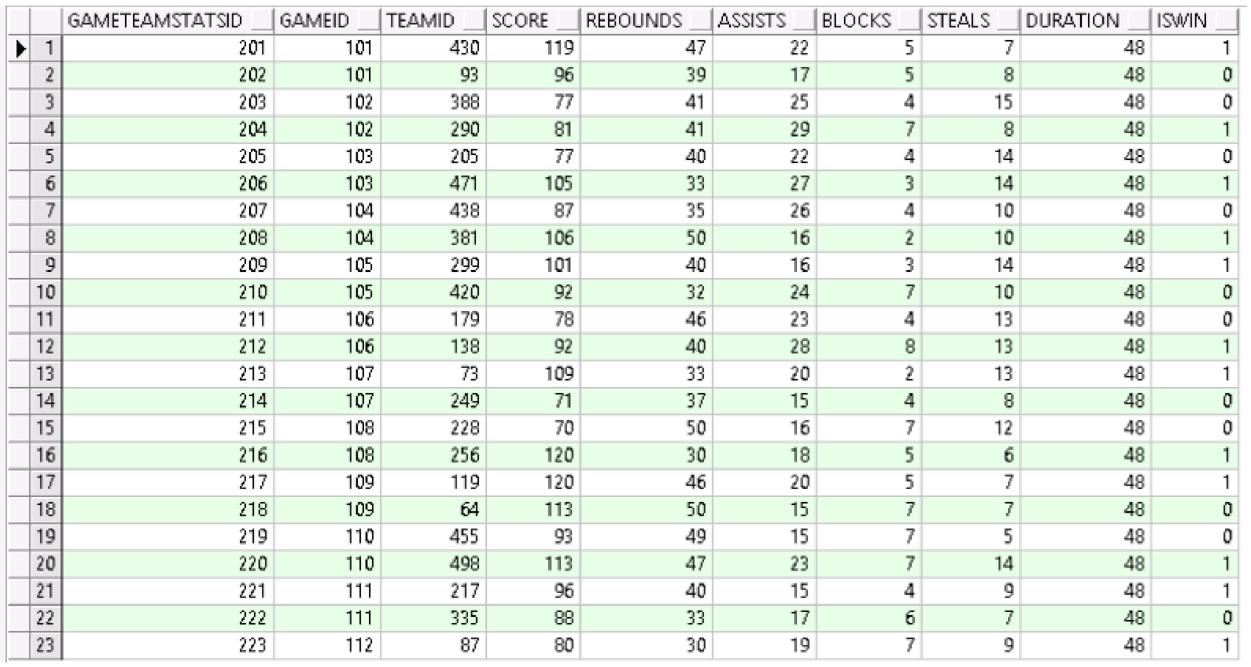
Field:

Fieldtype:

SQL function:

additional Oracle processing, for example: sub

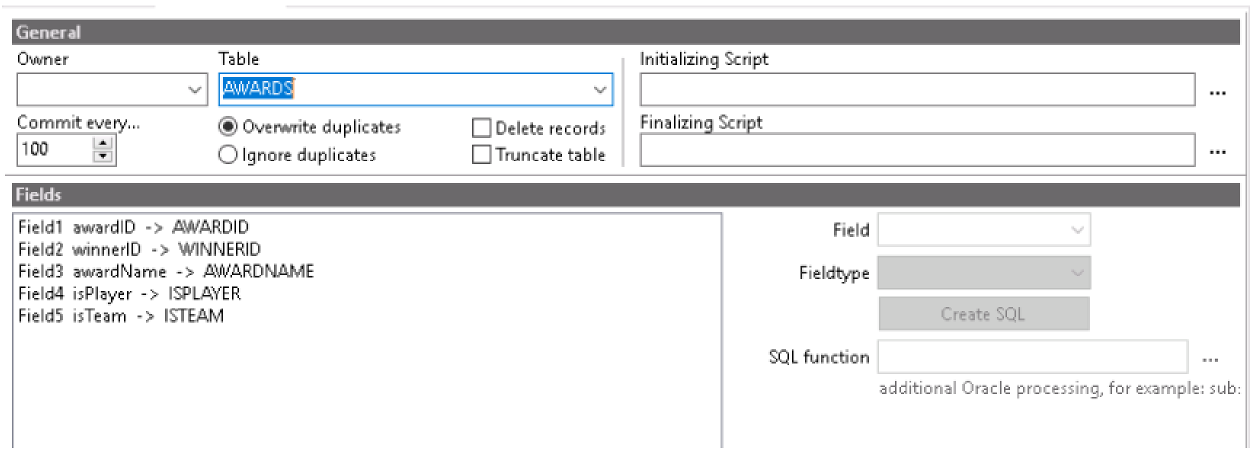
<u>Result</u>



	GAMETEAMSTATSID	GAMEID	TEAMID	SCORE	REBOUNDS	ASSISTS	BLOCKS	STEALS	DURATION	ISWIN
1	201	101	430	119	47	22	5	7	48	1
2	202	101	93	96	39	17	5	8	48	0
3	203	102	388	77	41	25	4	15	48	0
4	204	102	290	81	41	29	7	8	48	1
5	205	103	205	77	40	22	4	14	48	0
6	206	103	471	105	33	27	3	14	48	1
7	207	104	438	87	35	26	4	10	48	0
8	208	104	381	106	50	16	2	10	48	1
9	209	105	299	101	40	16	3	14	48	1
10	210	105	420	92	32	24	7	10	48	0
11	211	106	179	78	46	23	4	13	48	0
12	212	106	138	92	40	28	8	13	48	1
13	213	107	73	109	33	20	2	13	48	1
14	214	107	249	71	37	15	4	8	48	0
15	215	108	228	70	50	16	7	12	48	0
16	216	108	256	120	30	18	5	6	48	1
17	217	109	119	120	46	20	5	7	48	1
18	218	109	64	113	50	15	7	7	48	0
19	219	110	455	93	49	15	7	5	48	0
20	220	110	498	113	47	23	7	14	48	1
21	221	111	217	96	40	15	4	9	48	1
22	222	111	335	88	33	17	6	7	48	0
23	223	112	87	80	30	19	7	9	48	1

Inserting data in the Awards table, using the Text-Importer of PLSQL.

The configuration of the data:



General

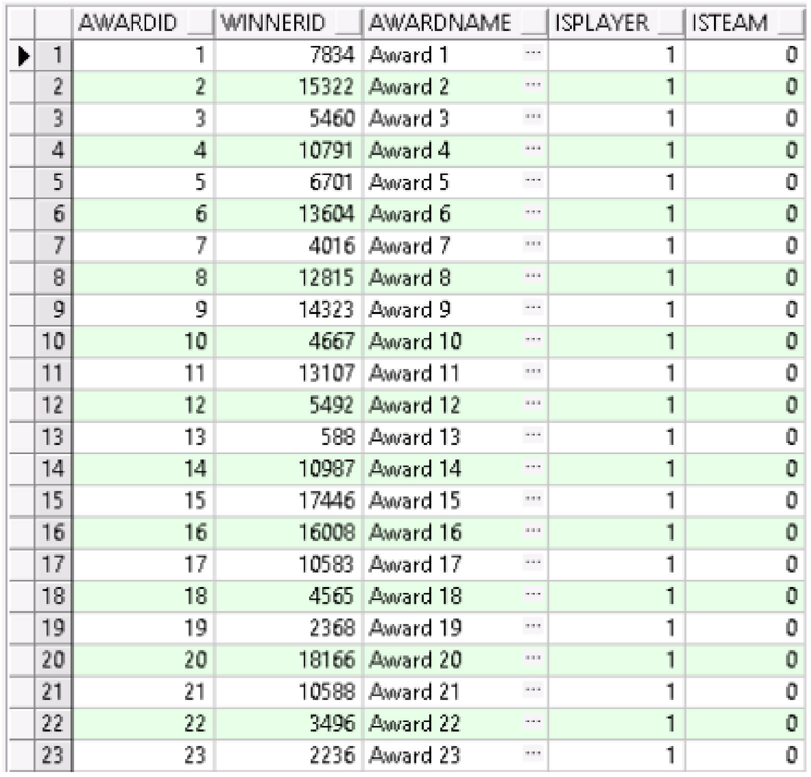
Owner:
Table: AWARDS
Commit every... 100
[✓ Overwrite duplicates] Ignore duplicates
☐ Delete records
☐ Truncate table
Initializing Script:
Finalizing Script:

Fields

```
Field1  awardID  -> AWARDID
Field2  winnerID  -> WINNERID
Field3  awardName  -> AWARDNAME
Field4  isPlayer  -> ISPLAYER
Field5  isTeam  -> ISTEAM
```

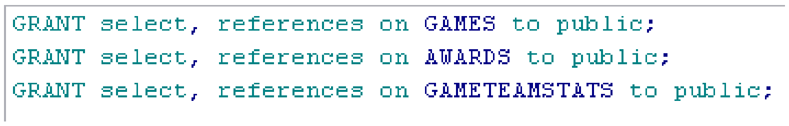
Field:
Fieldtype:
SQL function:
additional Oracle processing, for example: sub:

Result



	AWARDID	WINNERID	AWARDNAME	ISPLAYER	ISTEAM
1	1	7834	Award 1	1	0
2	2	15322	Award 2	1	0
3	3	5460	Award 3	1	0
4	4	10791	Award 4	1	0
5	5	6701	Award 5	1	0
6	6	13604	Award 6	1	0
7	7	4016	Award 7	1	0
8	8	12815	Award 8	1	0
9	9	14323	Award 9	1	0
10	10	4667	Award 10	1	0
11	11	13107	Award 11	1	0
12	12	5492	Award 12	1	0
13	13	588	Award 13	1	0
14	14	10987	Award 14	1	0
15	15	17446	Award 15	1	0
16	16	16008	Award 16	1	0
17	17	10583	Award 17	1	0
18	18	4565	Award 18	1	0
19	19	2368	Award 19	1	0
20	20	18166	Award 20	1	0
21	21	10588	Award 21	1	0
22	22	3496	Award 22	1	0
23	23	2236	Award 23	1	0

Grant Tables



```
GRANT select, references on GAMES to public;
GRANT select, references on AWARDS to public;
GRANT select, references on GAMETEAMSTATS to public;
```

Eight queries

1 -Query

This query retrieves the total number of games played by each team.

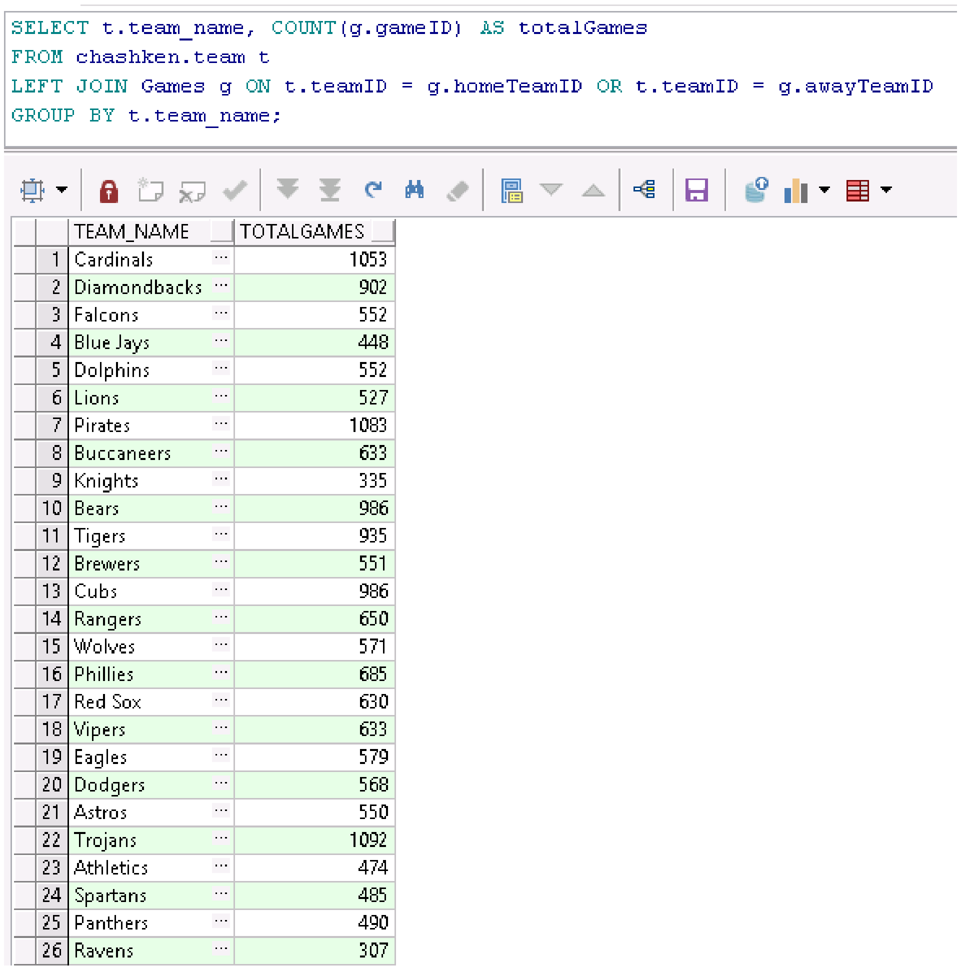
SQL Code

```
SELECT t.teamName, COUNT(g.gameID) AS totalGames
FROM chashken.team t
LEFT JOIN Games g ON t.teamID = g.homeTeamID OR t.teamID = g.awayTeamID
GROUP BY t.teamName;
```

Motivation

The motivation behind this query is to obtain the total number of games played by each team in a season. By joining the "Games" table with the "chashken.team" table and using the COUNT function, we can determine the game count for each team.

Result



```
SELECT t.team_name, COUNT(g.gameID) AS totalGames
FROM chashken.team t
LEFT JOIN Games g ON t.teamID = g.homeTeamID OR t.teamID = g.awayTeamID
GROUP BY t.team_name;
```

	TEAM_NAME	TOTALGAMES
1	Cardinals	1053
2	Diamondbacks	902
3	Falcons	552
4	Blue Jays	448
5	Dolphins	552
6	Lions	527
7	Pirates	1083
8	Buccaneers	633
9	Knights	335
10	Bears	986
11	Tigers	935
12	Brewers	551
13	Cubs	986
14	Rangers	650
15	Wolves	571
16	Phillies	685
17	Red Sox	630
18	Vipers	633
19	Eagles	579
20	Dodgers	568
21	Astros	550
22	Trojans	1092
23	Athletics	474
24	Spartans	485
25	Panthers	490
26	Ravens	307

2 -Query

This query retrieves the average score for each team in their home games.

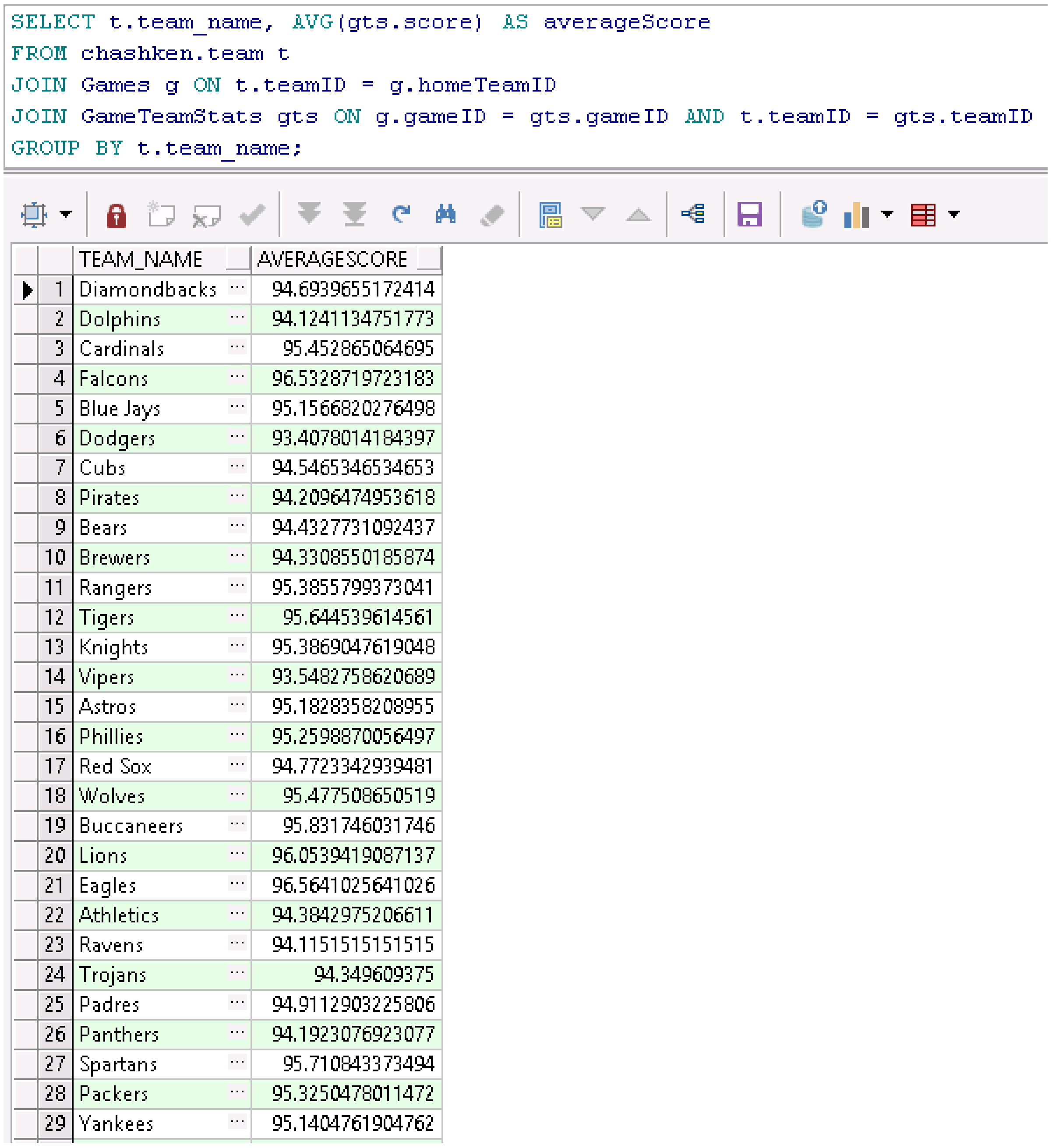
SQL Code

```sql
SELECT t.teamName, AVG(gts.score) AS averageScore
FROM chashken.team t
JOIN Games g ON t.teamID = g.homeTeamID
JOIN GameTeamStats gts ON g.gameID = gts.gameID AND t.teamID = gts.teamID
GROUP BY t.teamName;
```

Motivation

The motivation behind this query is to calculate the average score for each team in their home games. By joining the necessary tables and using the AVG function, we can determine the average score achieved by each team when playing at home.

Result



```sql
SELECT t.team_name, AVG(gts.score) AS averageScore
FROM chashken.team t
JOIN Games g ON t.teamID = g.homeTeamID
JOIN GameTeamStats gts ON g.gameID = gts.gameID AND t.teamID = gts.teamID
GROUP BY t.team_name;
```

	TEAM_NAME	AVERAGESCORE
1	Diamondbacks	94.6939655172414
2	Dolphins	94.1241134751773
3	Cardinals	95.452865064695
4	Falcons	96.5328719723183
5	Blue Jays	95.1566820276498
6	Dodgers	93.4078014184397
7	Cubs	94.5465346534653
8	Pirates	94.2096474953618
9	Bears	94.4327731092437
10	Brewers	94.3308550185874
11	Rangers	95.3855799373041
12	Tigers	95.644539614561
13	Knights	95.3869047619048
14	Vipers	93.5482758620689
15	Astros	95.1828358208955
16	Phillies	95.2598870056497
17	Red Sox	94.7723342939481
18	Wolves	95.477508650519
19	Buccaneers	95.831746031746
20	Lions	96.0539419087137
21	Eagles	96.5641025641026
22	Athletics	94.3842975206611
23	Ravens	94.1151515151515
24	Trojans	94.349609375
25	Padres	94.9112903225806
26	Panthers	94.1923076923077
27	Spartans	95.710843373494
28	Packers	95.3250478011472
29	Yankees	95.1404761904762

3 -Query

This query retrieves all the records from the "Games" table where the game date and time fall within the specified range.

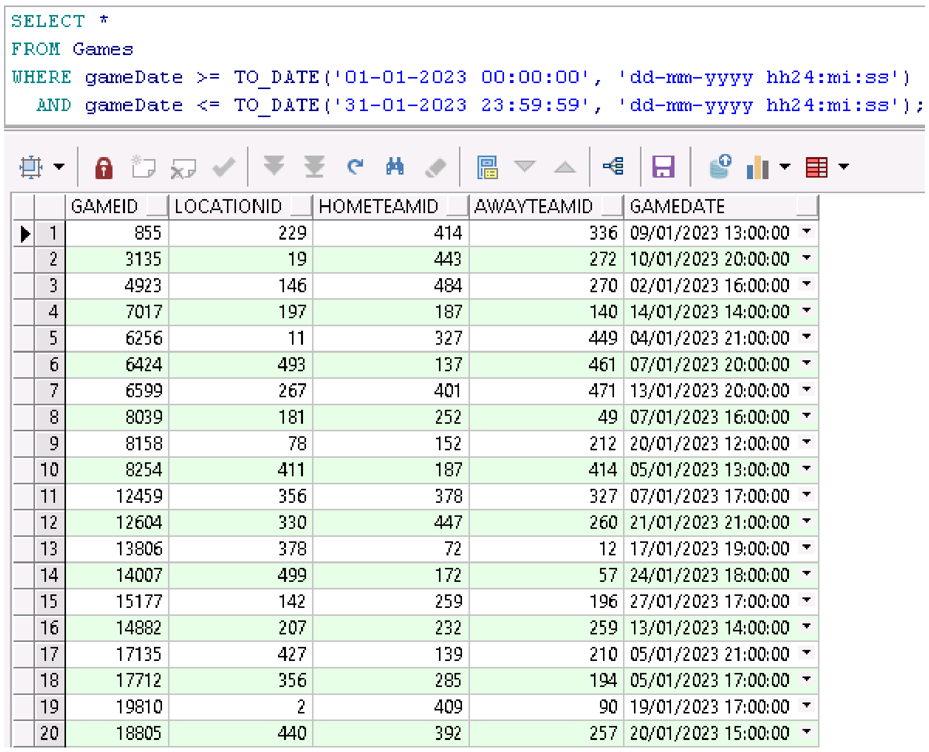
SQL Code

```
SELECT *
FROM Games
WHERE gameDateTime >= TO_DATE('01-01-2023 00:00:00', 'dd-mm-yyyy hh24:mi:ss')
  AND gameDateTime <= TO_DATE('31-01-2023 23:59:59', 'dd-mm-yyyy hh24:mi:ss');
```

Motivation

The motivation behind this query is to obtain the records from the "Games" table that occurred within a specific date and time range. By using the **>=** and **<=** operators along with the **TO_DATE** function to convert the given date and time strings into the appropriate format, we can filter the results to include only the games that took place between January 1, 2023, at 00:00:00 and January 31, 2023, at 23:59:59. This query helps in retrieving a subset of game records for a particular timeframe, which can be useful for analysis or reporting purposes.

Result



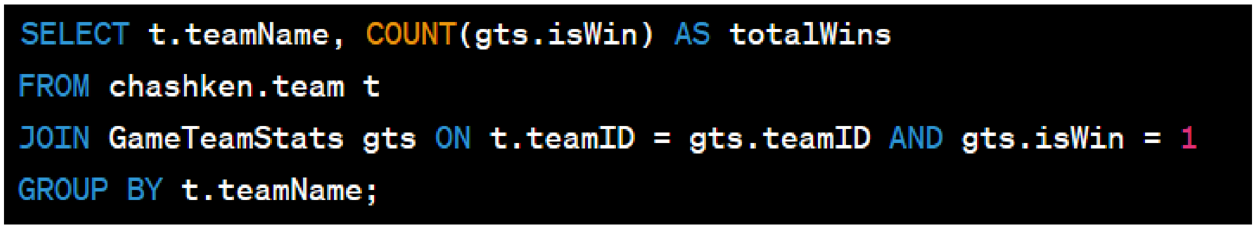
```
SELECT *
FROM Games
WHERE gameDate >= TO_DATE('01-01-2023 00:00:00', 'dd-mm-yyyy hh24:mi:ss')
  AND gameDate <= TO_DATE('31-01-2023 23:59:59', 'dd-mm-yyyy hh24:mi:ss');
```

	GAMEID	LOCATIONID	HOMETEAMID	AWAYTEAMID	GAMEDATE
1	855	229	414	336	09/01/2023 13:00:00
2	3135	19	443	272	10/01/2023 20:00:00
3	4923	146	484	270	02/01/2023 16:00:00
4	7017	197	187	140	14/01/2023 14:00:00
5	6256	11	327	449	04/01/2023 21:00:00
6	6424	493	137	461	07/01/2023 20:00:00
7	6599	267	401	471	13/01/2023 20:00:00
8	8039	181	252	49	07/01/2023 16:00:00
9	8158	78	152	212	20/01/2023 12:00:00
10	8254	411	187	414	05/01/2023 13:00:00
11	12459	356	378	327	07/01/2023 17:00:00
12	12604	330	447	260	21/01/2023 21:00:00
13	13806	378	72	12	17/01/2023 19:00:00
14	14007	499	172	57	24/01/2023 18:00:00
15	15177	142	259	196	27/01/2023 17:00:00
16	14882	207	232	259	13/01/2023 14:00:00
17	17135	427	139	210	05/01/2023 21:00:00
18	17712	356	285	194	05/01/2023 17:00:00
19	19810	2	409	90	19/01/2023 17:00:00
20	18805	440	392	257	20/01/2023 15:00:00

4 -Query

This query retrieves the number of games won by each team in a specific season.

SQL Code

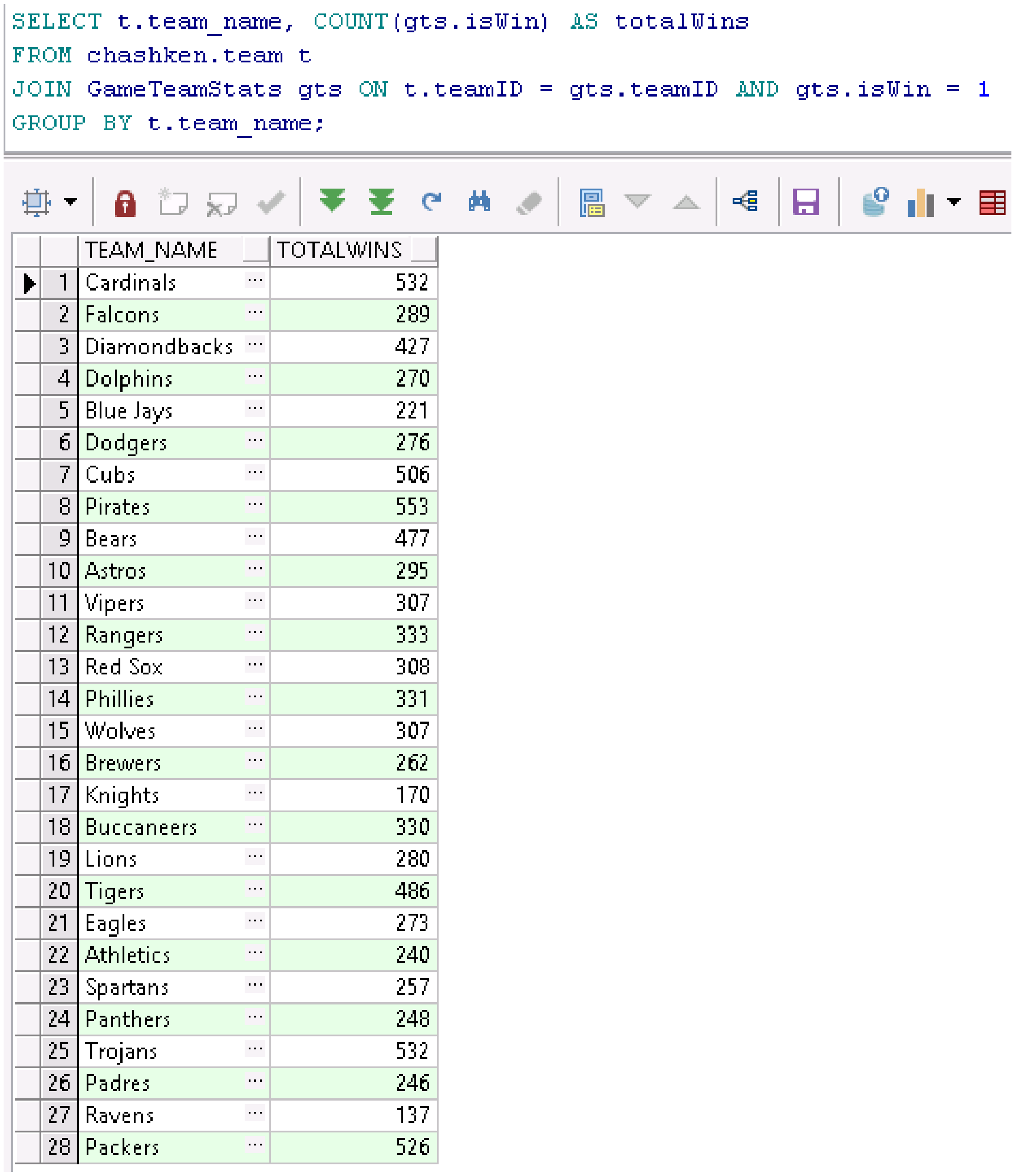


```sql
SELECT t.teamName, COUNT(gts.isWin) AS totalWins
FROM chashken.team t
JOIN GameTeamStats gts ON t.teamID = gts.teamID AND gts.isWin = 1
GROUP BY t.teamName;
```

Motivation

The motivation behind this query is to determine the number of games won by each team in a specific season. By joining the "chashken.team" table with the "GameTeamStats" table and filtering for wins (isWin = 1), we can count the total number of wins for each team.

Result



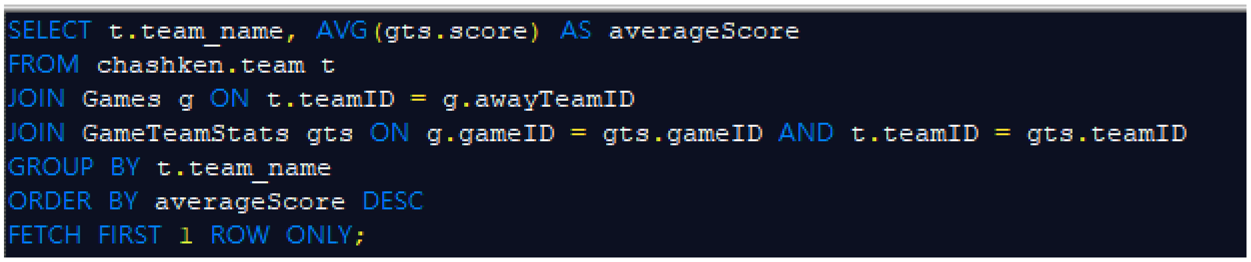
```sql
SELECT t.team_name, COUNT(gts.isWin) AS totalWins
FROM chashken.team t
JOIN GameTeamStats gts ON t.teamID = gts.teamID AND gts.isWin = 1
GROUP BY t.team_name;
```

	TEAM_NAME	TOTALWINS
1	Cardinals	532
2	Falcons	289
3	Diamondbacks	427
4	Dolphins	270
5	Blue Jays	221
6	Dodgers	276
7	Cubs	506
8	Pirates	553
9	Bears	477
10	Astros	295
11	Vipers	307
12	Rangers	333
13	Red Sox	308
14	Phillies	331
15	Wolves	307
16	Brewers	262
17	Knights	170
18	Buccaneers	330
19	Lions	280
20	Tigers	486
21	Eagles	273
22	Athletics	240
23	Spartans	257
24	Panthers	248
25	Trojans	532
26	Padres	246
27	Ravens	137
28	Packers	526

5 -Query

This query retrieves the team with the highest average score in away games.

SQL Code

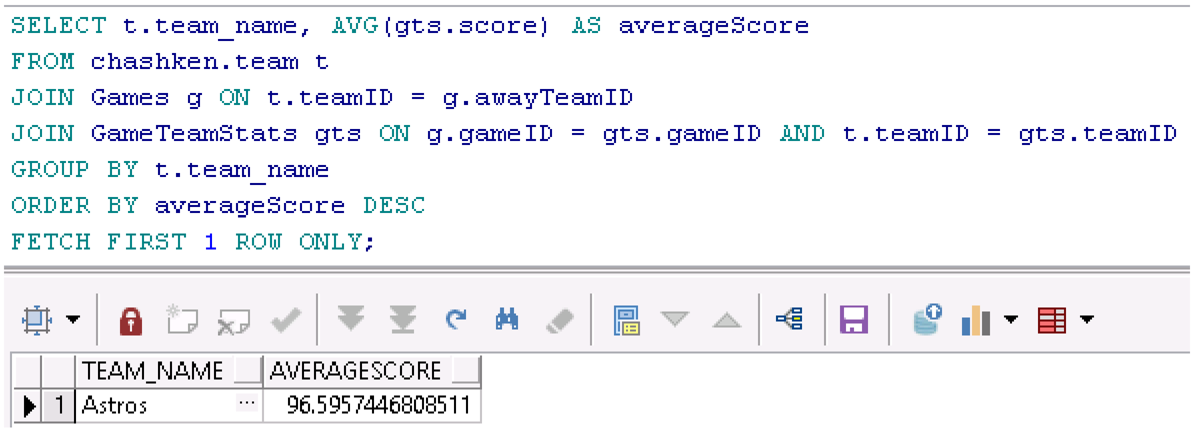


```
SELECT t.team_name, AVG(gts.score) AS averageScore
FROM chashken.team t
JOIN Games g ON t.teamID = g.awayTeamID
JOIN GameTeamStats gts ON g.gameID = gts.gameID AND t.teamID = gts.teamID
GROUP BY t.team_name
ORDER BY averageScore DESC
FETCH FIRST 1 ROW ONLY;
```

Motivation

The motivation behind this query is to identify the team with the highest average score in away games. By joining the necessary tables, calculating the average score using AVG, and sorting the results in descending order, we can determine the team with the highest average score away from home.

Result



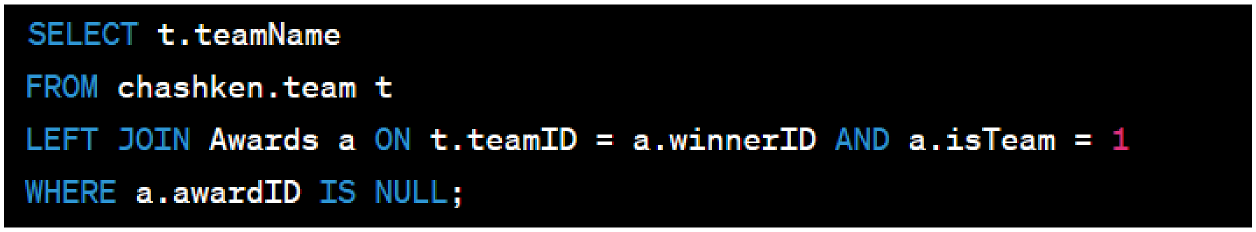
```
SELECT t.team_name, AVG(gts.score) AS averageScore
FROM chashken.team t
JOIN Games g ON t.teamID = g.awayTeamID
JOIN GameTeamStats gts ON g.gameID = gts.gameID AND t.teamID = gts.teamID
GROUP BY t.team_name
ORDER BY averageScore DESC
FETCH FIRST 1 ROW ONLY;
```

	TEAM_NAME	AVERAGESCORE
1	Astros	96.5957446808511

6 -Query

This query retrieves the teams that have never won an award.

SQL Code



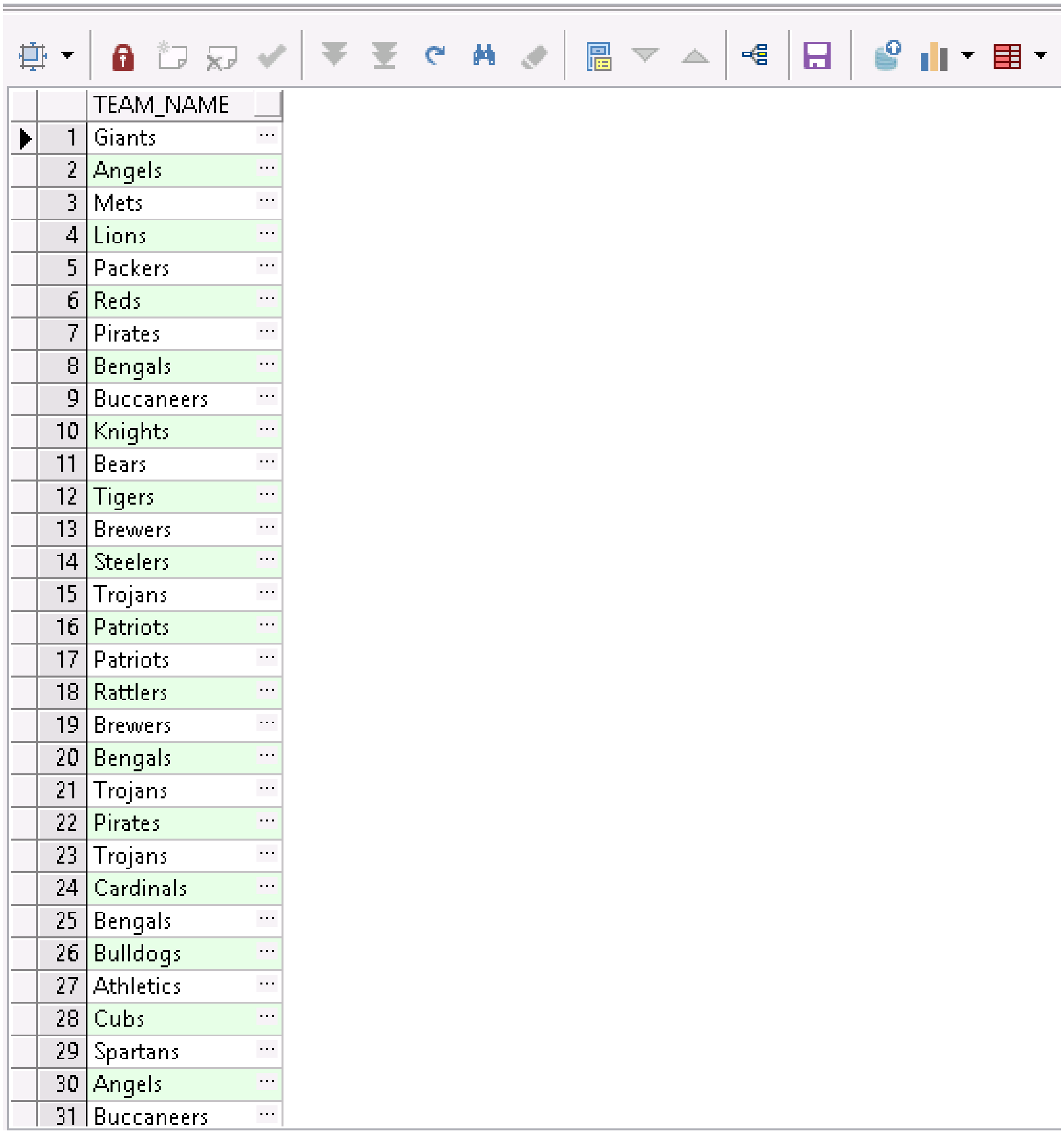
```
SELECT t.teamName
FROM chashken.team t
LEFT JOIN Awards a ON t.teamID = a.winnerID AND a.isTeam = 1
WHERE a.awardID IS NULL;
```

Motivation

The motivation behind this query is to identify the teams that have never won an award. By joining the "chashken.team" table with the "Awards" table, filtering for team awards (isTeam = 1), and selecting the teams with no corresponding awards (awardID IS NULL), we can determine the teams without any accolades.

Result

```
SELECT t.team_name
FROM chashken.team t
LEFT JOIN Awards a ON t.teamID = a.winnerID AND a.isTeam = 1
WHERE a.awardID IS NULL;
```



	TEAM_NAME
1	Giants
2	Angels
3	Mets
4	Lions
5	Packers
6	Reds
7	Pirates
8	Bengals
9	Buccaneers
10	Knights
11	Bears
12	Tigers
13	Brewers
14	Steelers
15	Trojans
16	Patriots
17	Patriots
18	Rattlers
19	Brewers
20	Bengals
21	Trojans
22	Pirates
23	Trojans
24	Cardinals
25	Bengals
26	Bulldogs
27	Athletics
28	Cubs
29	Spartans
30	Angels
31	Buccaneers

7 -Query

The query retrieves the top 10 teams with the highest points, including the date and total points.

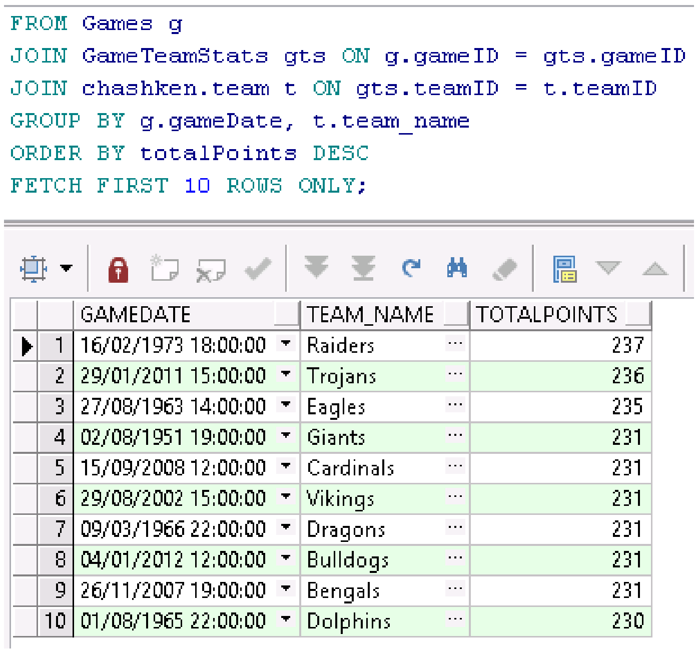
SQL Code

```
SELECT g.gameDate, t.teamName, SUM(gts.score) AS totalPoints
FROM Games g
JOIN GameTeamStats gts ON g.gameID = gts.gameID
JOIN chashken.team t ON gts.teamID = t.teamID
GROUP BY g.gameDate, t.teamName
ORDER BY totalPoints DESC
FETCH FIRST 10 ROWS ONLY;
```

Motivation

The motivation behind this query is to identify the top 10 teams with the highest total points scored across all games, along with the corresponding date of each game. By joining the necessary tables (`Games`, `GameTeamStats`, and `chashken.team`), summing the scores using the `SUM` function, grouping the results by game date and team name, and sorting the results in descending order based on the total points, we can determine the top-performing teams. The `FETCH FIRST 10 ROWS ONLY` clause limits the result set to only the top 10 rows, providing a concise view of the teams with the highest total points scored. This query allows for analysis and comparison of team performances based on their overall scoring abilities.

Result



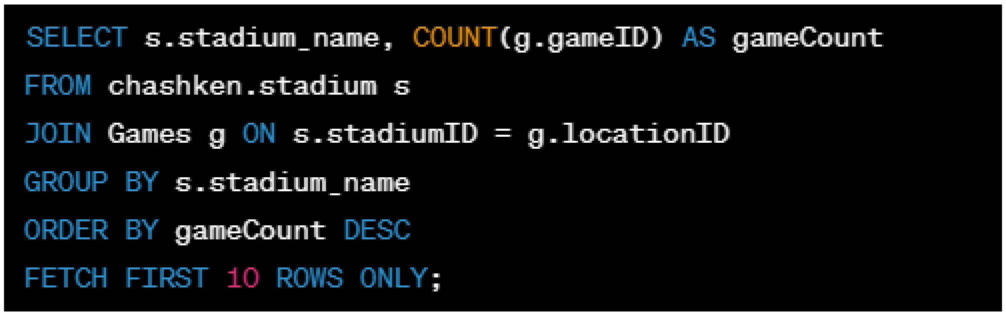
```
FROM Games g
JOIN GameTeamStats gts ON g.gameID = gts.gameID
JOIN chashken.team t ON gts.teamID = t.teamID
GROUP BY g.gameDate, t.team_name
ORDER BY totalPoints DESC
FETCH FIRST 10 ROWS ONLY;
```

	GAMEDATE	TEAM_NAME	TOTALPOINTS
1	16/02/1973 18:00:00	Raiders	237
2	29/01/2011 15:00:00	Trojans	236
3	27/08/1963 14:00:00	Eagles	235
4	02/08/1951 19:00:00	Giants	231
5	15/09/2008 12:00:00	Cardinals	231
6	29/08/2002 15:00:00	Vikings	231
7	09/03/1966 22:00:00	Dragons	231
8	04/01/2012 12:00:00	Bulldogs	231
9	26/11/2007 19:00:00	Bengals	231
10	01/08/1965 22:00:00	Dolphins	230

8 -Query

This query retrieves the top 10 stadiums with the highest number of games played.

SQL Code

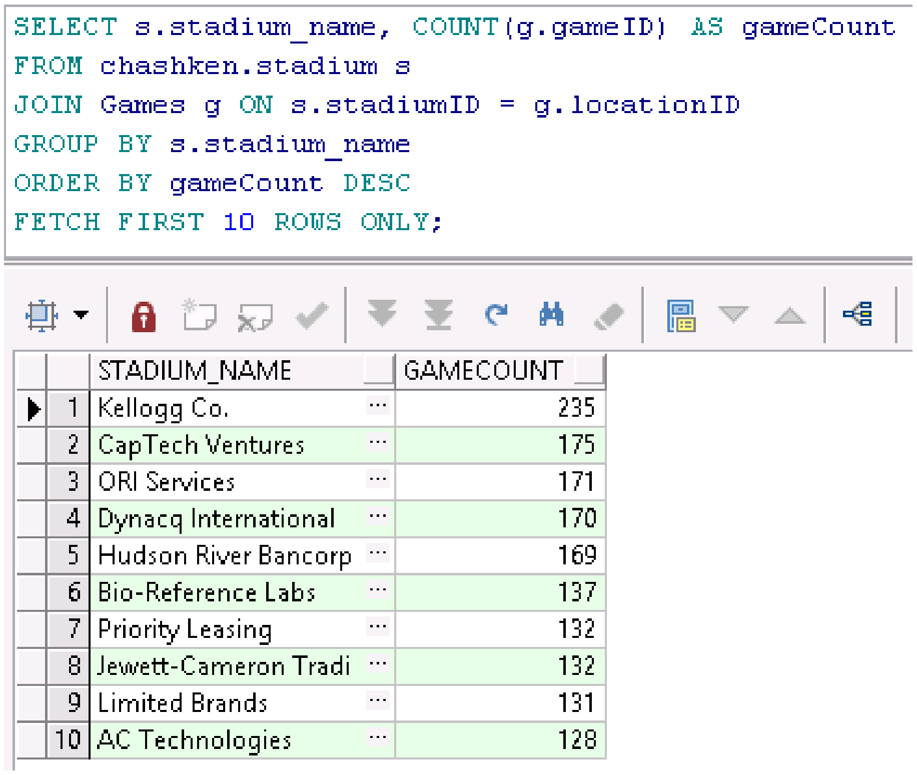


```sql
SELECT s.stadium_name, COUNT(g.gameID) AS gameCount
FROM chashken.stadium s
JOIN Games g ON s.stadiumID = g.locationID
GROUP BY s.stadium_name
ORDER BY gameCount DESC
FETCH FIRST 10 ROWS ONLY;
```

Motivation

The motivation behind this query is to identify the top 10 stadiums that have hosted the highest number of games.

Result



```sql
SELECT s.stadium_name, COUNT(g.gameID) AS gameCount
FROM chashken.stadium s
JOIN Games g ON s.stadiumID = g.locationID
GROUP BY s.stadium_name
ORDER BY gameCount DESC
FETCH FIRST 10 ROWS ONLY;
```

	STADIUM_NAME	GAMECOUNT
1	Kellogg Co.	235
2	CapTech Ventures	175
3	ORI Services	171
4	Dynacq International	170
5	Hudson River Bancorp	169
6	Bio-Reference Labs	137
7	Priority Leasing	132
8	Jewett-Cameron Tradi	132
9	Limited Brands	131
10	AC Technologies	128

Indexes

Speed Improvement

Index for Query -6

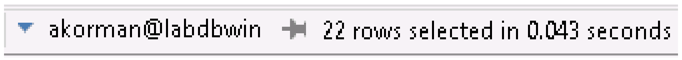
The Index:

```
CREATE INDEX idx_awards_winner ON Awards (winnerID, isTeam);
```

Motivation:

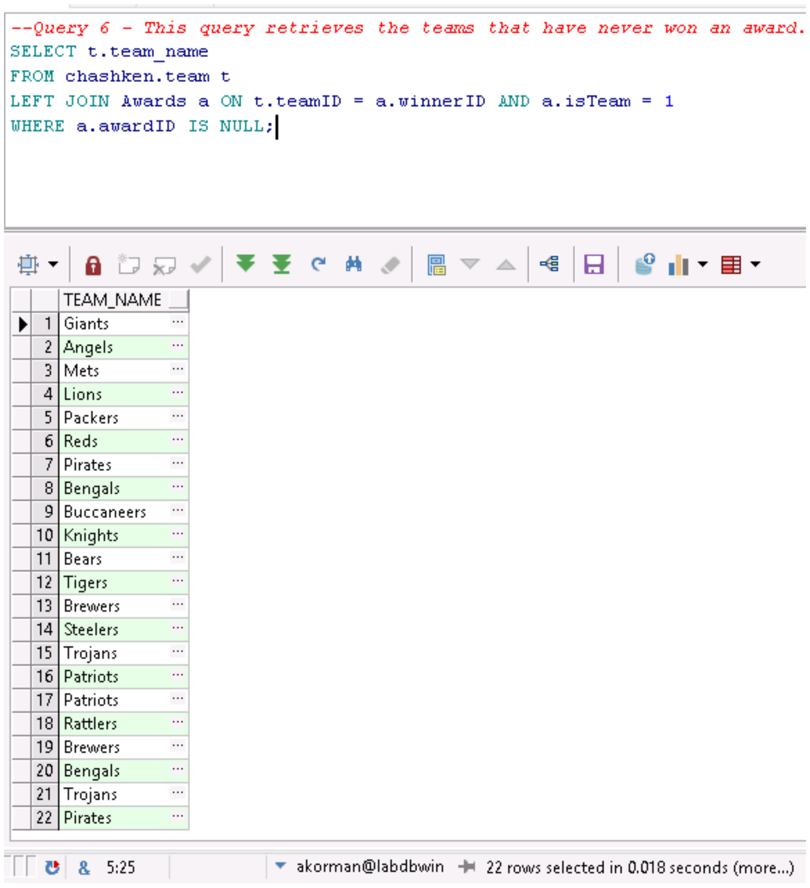
This index improves the performance of Query 6, which retrieves the teams that have never won an award. The index on the **winnerID** column allows for faster filtering and joining operations when searching for teams that have no corresponding records in the **Awards** table.

Time before: 0.043 sec



```
akorman@labdbwin    22 rows selected in 0.043 seconds
```

Time after: 0.018 sec



```
--Query 6 - This query retrieves the teams that have never won an award.
SELECT t.team_name
FROM chashken.team t
LEFT JOIN Awards a ON t.teamID = a.winnerID AND a.isTeam = 1
WHERE a.awardID IS NULL;
```

	TEAM_NAME
1	Giants
2	Angels
3	Mets
4	Lions
5	Packers
6	Reds
7	Pirates
8	Bengals
9	Buccaneers
10	Knights
11	Bears
12	Tigers
13	Brewers
14	Steelers
15	Trojans
16	Patriots
17	Patriots
18	Rattlers
19	Brewers
20	Bengals
21	Trojans
22	Pirates

```
5:25    akorman@labdbwin    22 rows selected in 0.018 seconds (more...)
```

Index for Query -3

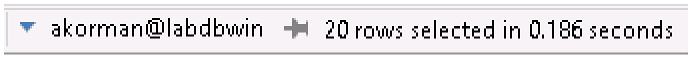
The Index:

```
CREATE INDEX idx_games_date ON Games (gameDate);
```

Motivation:

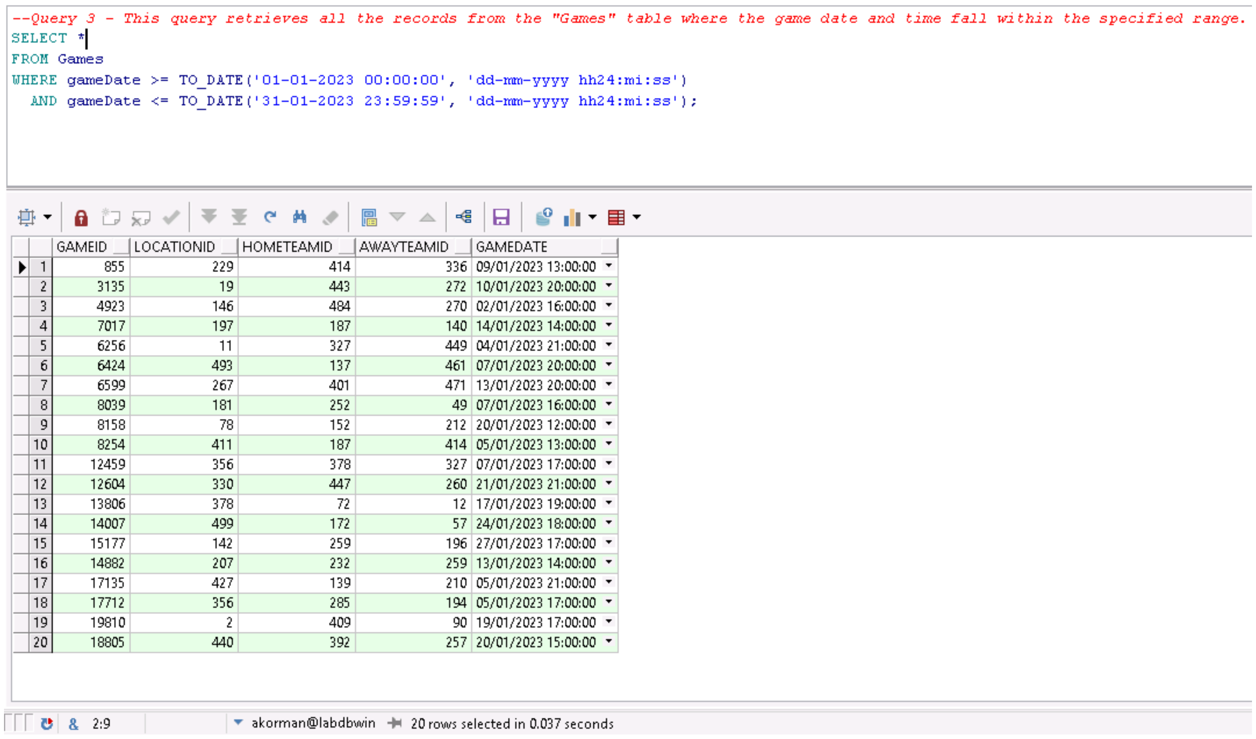
This index improves the performance of Query 3, which retrieves all records from the **Games** table within a specified date range. The index on the **gameDate** column speeds up the search operation, allowing for faster filtering and retrieval of the relevant game records based on the specified date range.

Time before: 0.186 sec



```
akorman@labdbwin   20 rows selected in 0.186 seconds
```

Time after: 0.037 sec



```
--Query 3 - This query retrieves all the records from the "Games" table where the game date and time fall within the specified range.
SELECT *
FROM Games
WHERE gameDate >= TO_DATE('01-01-2023 00:00:00', 'dd-mm-yyyy hh24:mi:ss')
  AND gameDate <= TO_DATE('31-01-2023 23:59:59', 'dd-mm-yyyy hh24:mi:ss');
```

	GAMEID	LOCATIONID	HOMETEAMID	AWAYTEAMID	GAMEDATE
1	855	229	414	336	09/01/2023 13:00:00
2	3135	19	443	272	10/01/2023 20:00:00
3	4923	146	484	270	02/01/2023 16:00:00
4	7017	197	187	140	14/01/2023 14:00:00
5	6256	11	327	449	04/01/2023 21:00:00
6	6424	493	137	461	07/01/2023 20:00:00
7	6599	267	401	471	13/01/2023 20:00:00
8	8039	181	252	49	07/01/2023 16:00:00
9	8158	78	152	212	20/01/2023 12:00:00
10	8254	411	187	414	05/01/2023 13:00:00
11	12459	356	378	327	07/01/2023 17:00:00
12	12604	330	447	260	21/01/2023 21:00:00
13	13806	378	72	12	17/01/2023 19:00:00
14	14007	499	172	57	24/01/2023 18:00:00
15	15177	142	259	196	27/01/2023 17:00:00
16	14882	207	232	259	13/01/2023 14:00:00
17	17135	427	139	210	05/01/2023 21:00:00
18	17712	356	285	194	05/01/2023 17:00:00
19	19810	2	409	90	19/01/2023 17:00:00
20	18805	440	392	257	20/01/2023 15:00:00

```
2:9   akorman@labdbwin   20 rows selected in 0.037 seconds
```

Index for Query -7

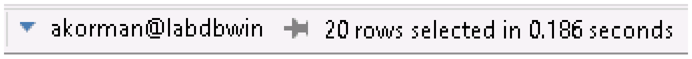
The Index:

```
CREATE INDEX idx_gamestats_team_game ON GameTeamStats (teamID, gameID);
```

Motivation:

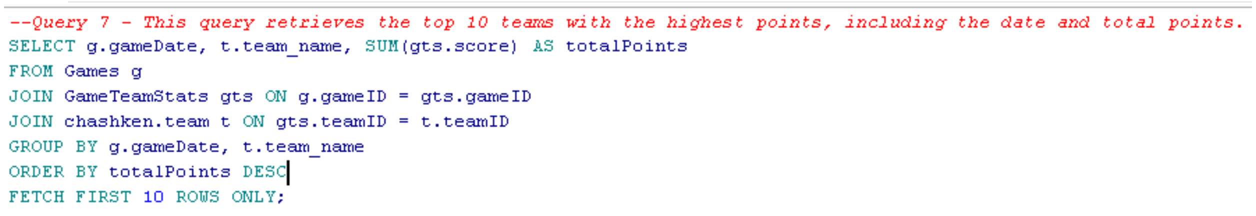
This index enhances the performance of Query 7, which retrieves the top 10 teams with the highest points, including the date and total points. The index on the **teamID** and **gameID** columns facilitates faster joins and aggregations between the **GameTeamStats**, **Games**, and **chashken.team** tables, improving the overall query execution time.

Time before: 0.186 sec

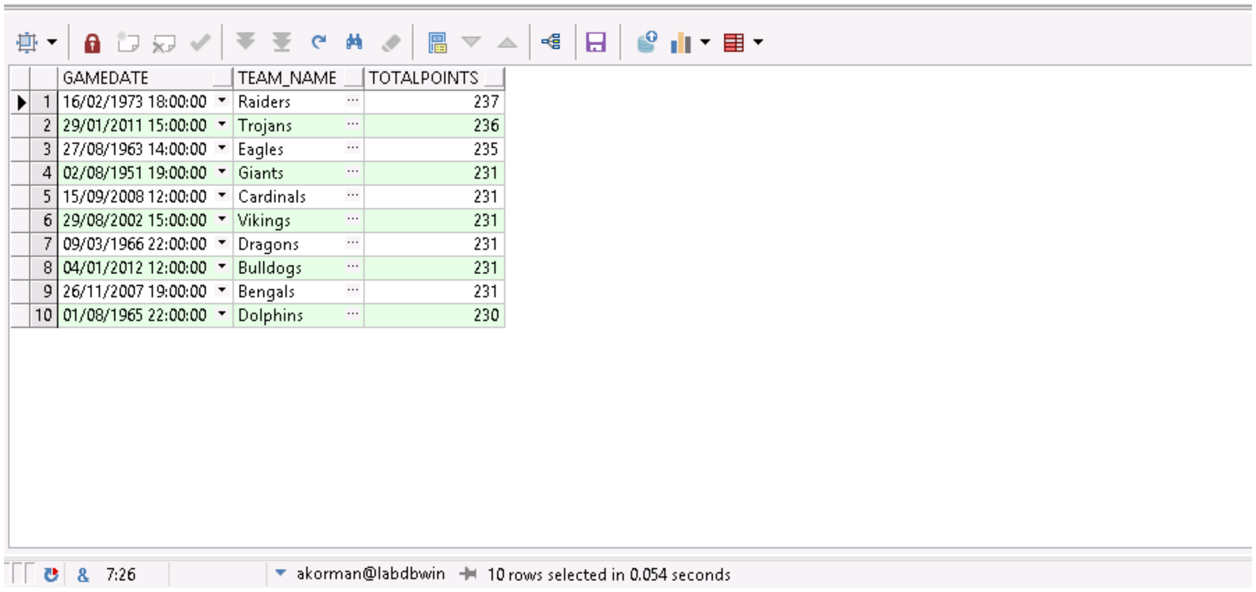


```
akorman@labdbwin   20 rows selected in 0.186 seconds
```

Time after: 0.054 sec



```
--Query 7 - This query retrieves the top 10 teams with the highest points, including the date and total points.
SELECT g.gameDate, t.team_name, SUM(gts.score) AS totalPoints
FROM Games g
JOIN GameTeamStats gts ON g.gameID = gts.gameID
JOIN chashken.team t ON gts.teamID = t.teamID
GROUP BY g.gameDate, t.team_name
ORDER BY totalPoints DESC
FETCH FIRST 10 ROWS ONLY;
```



	GAMEDATE	TEAM_NAME	TOTALPOINTS
1	16/02/1973 18:00:00	Raiders	237
2	29/01/2011 15:00:00	Trojans	236
3	27/08/1963 14:00:00	Eagles	235
4	02/08/1951 19:00:00	Giants	231
5	15/09/2008 12:00:00	Cardinals	231
6	29/08/2002 15:00:00	Vikings	231
7	09/03/1966 22:00:00	Dragons	231
8	04/01/2012 12:00:00	Bulldogs	231
9	26/11/2007 19:00:00	Bengals	231
10	01/08/1965 22:00:00	Dolphins	230

```
7:26   akorman@labdbwin   10 rows selected in 0.054 seconds
```

Views

1 -View

Upcoming Games

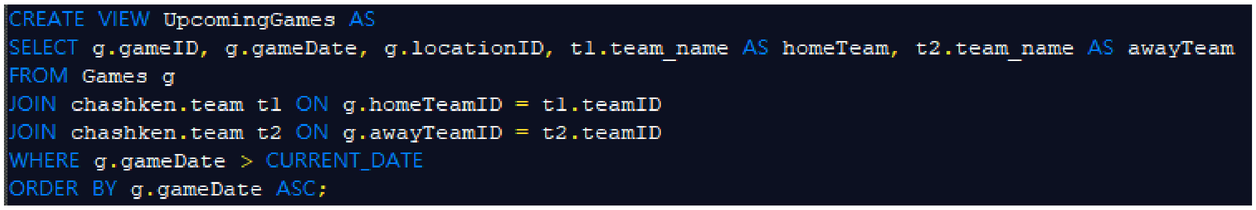
User Type:

This view will be helpful for regular users (General Audience).

Description:

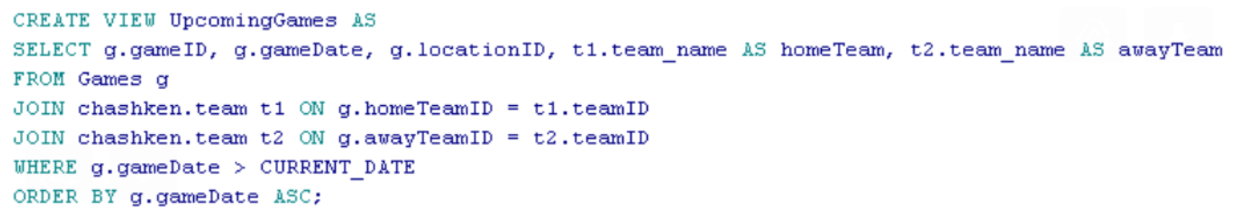
This view provides information about the upcoming games, including the teams, game date, and location. It allows regular users to stay updated on the upcoming matches.

Code:



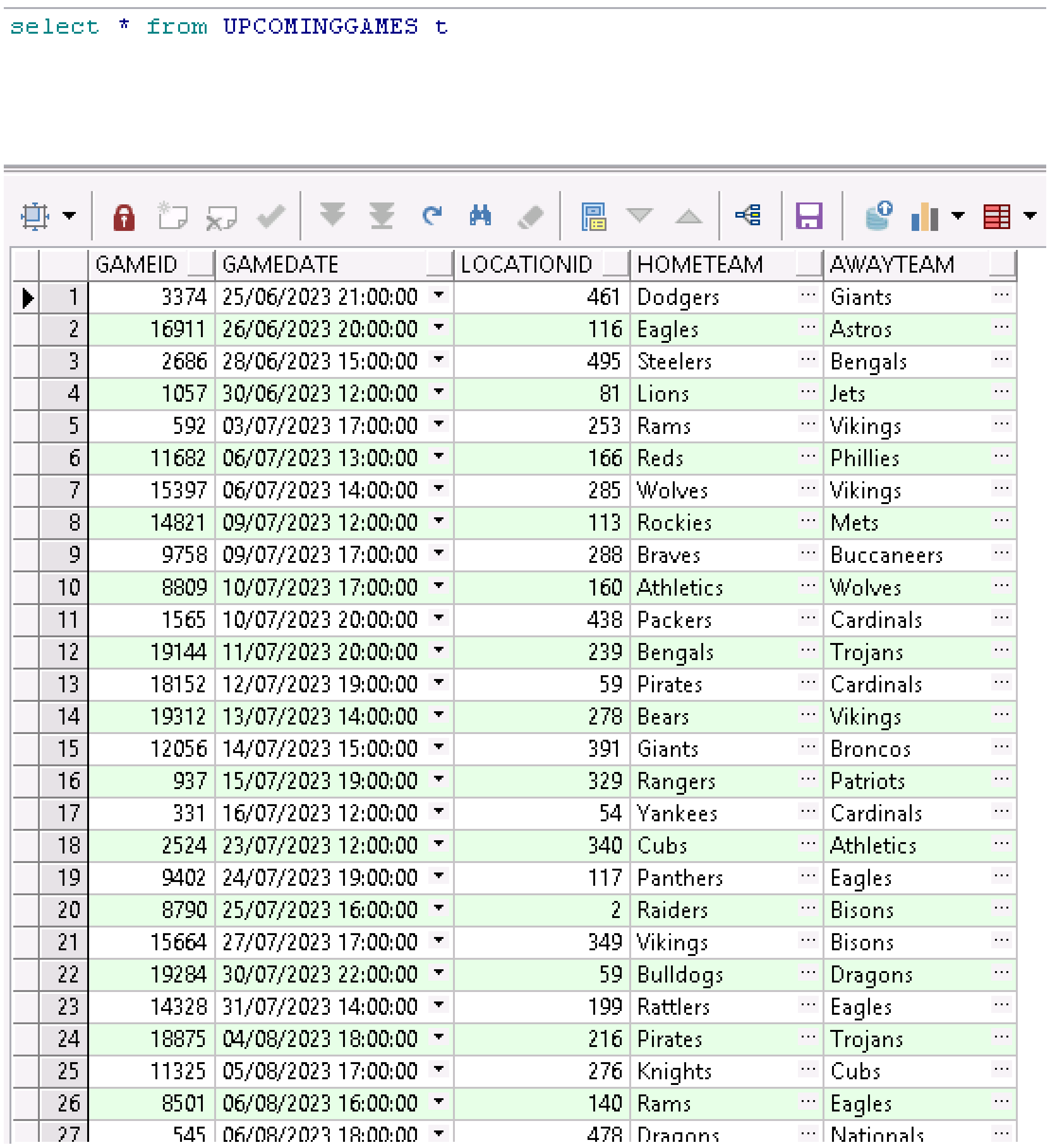
```
CREATE VIEW UpcomingGames AS
SELECT g.gameID, g.gameDate, g.locationID, t1.team_name AS homeTeam, t2.team_name AS awayTeam
FROM Games g
JOIN chashken.team t1 ON g.homeTeamID = t1.teamID
JOIN chashken.team t2 ON g.awayTeamID = t2.teamID
WHERE g.gameDate > CURRENT_DATE
ORDER BY g.gameDate ASC;
```

PLSQL:



```
CREATE VIEW UpcomingGames AS
SELECT g.gameID, g.gameDate, g.locationID, t1.team_name AS homeTeam, t2.team_name AS awayTeam
FROM Games g
JOIN chashken.team t1 ON g.homeTeamID = t1.teamID
JOIN chashken.team t2 ON g.awayTeamID = t2.teamID
WHERE g.gameDate > CURRENT_DATE
ORDER BY g.gameDate ASC;
```

Result:



```
select * from UPCOMINGGAMES t
```

	GAMEID	GAMEDATE	LOCATIONID	HOMETEAM	AWAYTEAM
1	3374	25/06/2023 21:00:00	461	Dodgers	Giants
2	16911	26/06/2023 20:00:00	116	Eagles	Astros
3	2686	28/06/2023 15:00:00	495	Steelers	Bengals
4	1057	30/06/2023 12:00:00	81	Lions	Jets
5	592	03/07/2023 17:00:00	253	Rams	Vikings
6	11682	06/07/2023 13:00:00	166	Reds	Phillies
7	15397	06/07/2023 14:00:00	285	Wolves	Vikings
8	14821	09/07/2023 12:00:00	113	Rockies	Mets
9	9758	09/07/2023 17:00:00	288	Braves	Buccaneers
10	8809	10/07/2023 17:00:00	160	Athletics	Wolves
11	1565	10/07/2023 20:00:00	438	Packers	Cardinals
12	19144	11/07/2023 20:00:00	239	Bengals	Trojans
13	18152	12/07/2023 19:00:00	59	Pirates	Cardinals
14	19312	13/07/2023 14:00:00	278	Bears	Vikings
15	12056	14/07/2023 15:00:00	391	Giants	Broncos
16	937	15/07/2023 19:00:00	329	Rangers	Patriots
17	331	16/07/2023 12:00:00	54	Yankees	Cardinals
18	2524	23/07/2023 12:00:00	340	Cubs	Athletics
19	9402	24/07/2023 19:00:00	117	Panthers	Eagles
20	8790	25/07/2023 16:00:00	2	Raiders	Bisons
21	15664	27/07/2023 17:00:00	349	Vikings	Bisons
22	19284	30/07/2023 22:00:00	59	Bulldogs	Dragons
23	14328	31/07/2023 14:00:00	199	Rattlers	Eagles
24	18875	04/08/2023 18:00:00	216	Pirates	Trojans
25	11325	05/08/2023 17:00:00	276	Knights	Cubs
26	8501	06/08/2023 16:00:00	140	Rams	Eagles
27	545	06/08/2023 18:00:00	478	Dragons	Nationals

2 -View

Team Standings

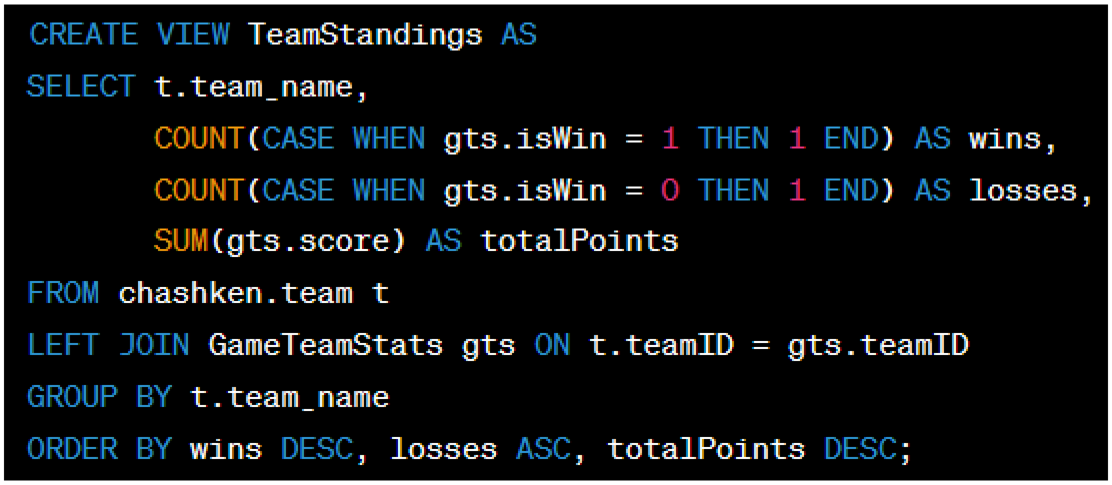
User Type:

This view will be helpful for regular users (General Audience).

Description:

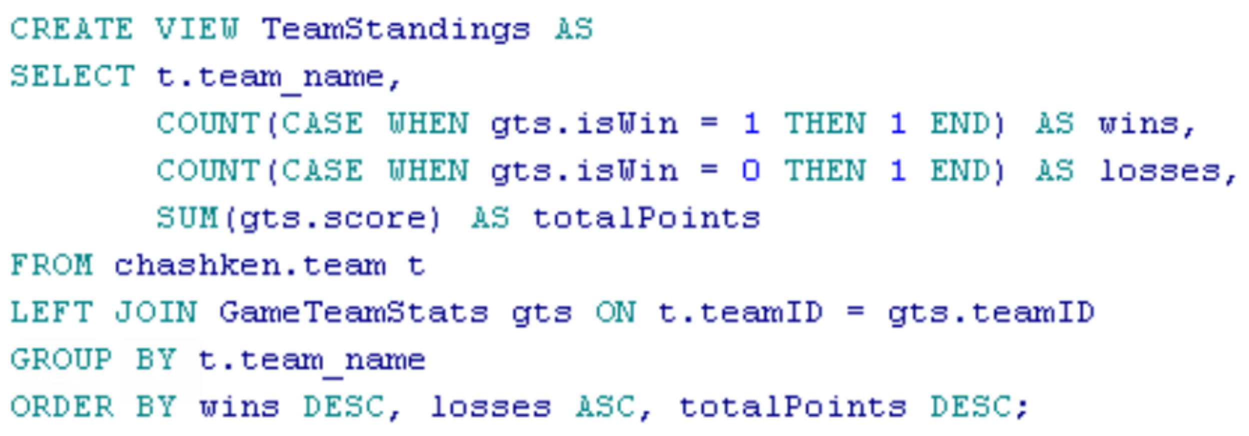
This view displays the current standings of all teams in the league. It includes team names, win-loss records, and points. It helps regular users track the performance and ranking of different teams.

Code:



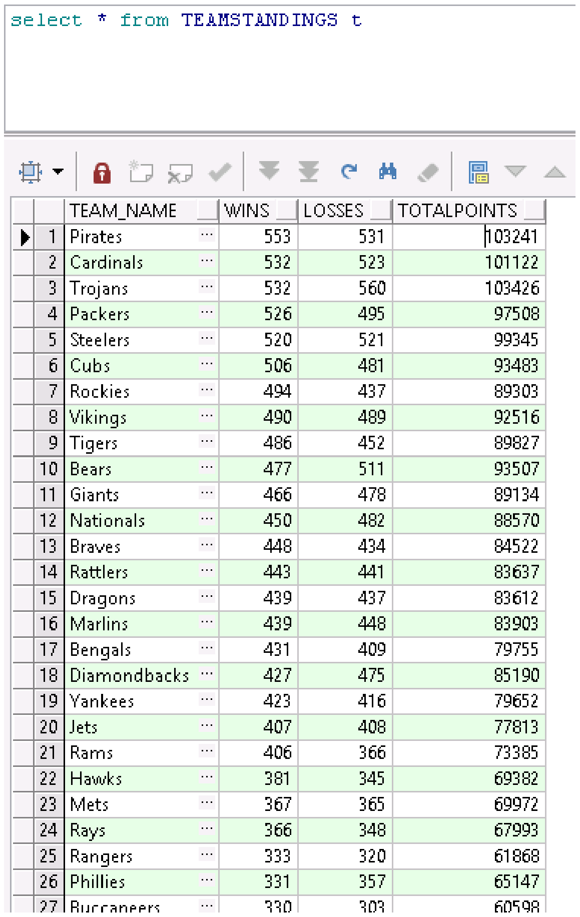
```
 CREATE VIEW TeamStandings AS
SELECT t.team_name,
       COUNT(CASE WHEN gts.isWin = 1 THEN 1 END) AS wins,
       COUNT(CASE WHEN gts.isWin = 0 THEN 1 END) AS losses,
       SUM(gts.score) AS totalPoints
FROM chashken.team t
LEFT JOIN GameTeamStats gts ON t.teamID = gts.teamID
GROUP BY t.team_name
ORDER BY wins DESC, losses ASC, totalPoints DESC;
```

PLSQL:



```
CREATE VIEW TeamStandings AS
SELECT t.team_name,
       COUNT(CASE WHEN gts.isWin = 1 THEN 1 END) AS wins,
       COUNT(CASE WHEN gts.isWin = 0 THEN 1 END) AS losses,
       SUM(gts.score) AS totalPoints
FROM chashken.team t
LEFT JOIN GameTeamStats gts ON t.teamID = gts.teamID
GROUP BY t.team_name
ORDER BY wins DESC, losses ASC, totalPoints DESC;
```

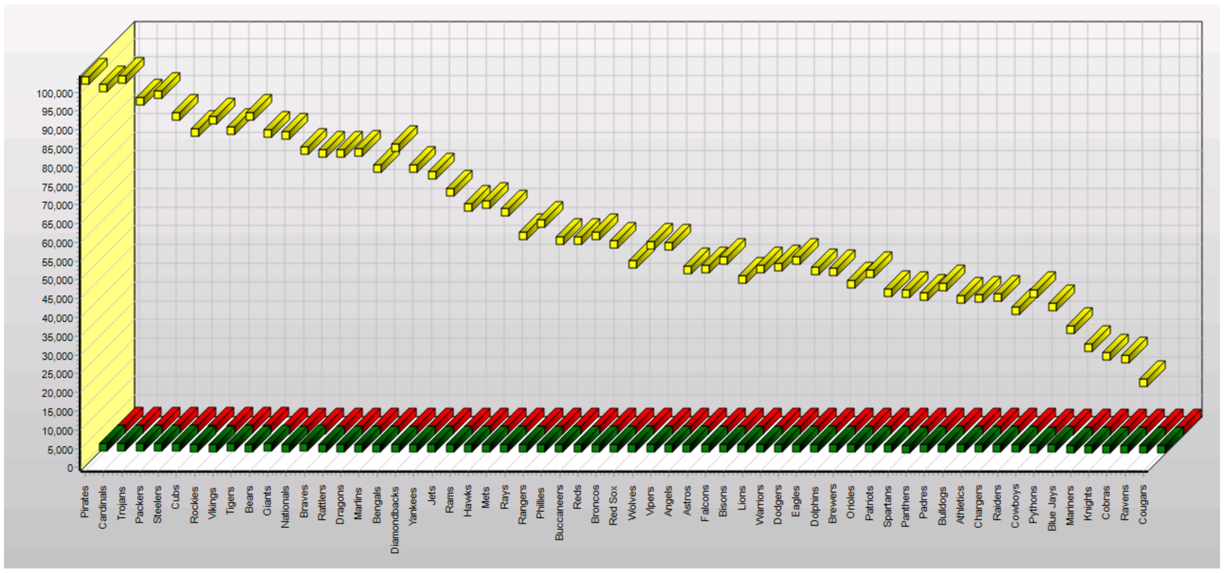
Result:



```
select * from TEAMSTANDINGS t
```

	TEAM_NAME	WINS	LOSSES	TOTALPOINTS
1	Pirates	553	531	103241
2	Cardinals	532	523	101122
3	Trojans	532	560	103426
4	Packers	526	495	97508
5	Steelers	520	521	99345
6	Cubs	506	481	93483
7	Rockies	494	437	89303
8	Vikings	490	489	92516
9	Tigers	486	452	89827
10	Bears	477	511	93507
11	Giants	466	478	89134
12	Nationals	450	482	88570
13	Braves	448	434	84522
14	Rattlers	443	441	83637
15	Dragons	439	437	83612
16	Marlins	439	448	83903
17	Bengals	431	409	79755
18	Diamondbacks	427	475	85190
19	Yankees	423	416	79652
20	Jets	407	408	77813
21	Rams	406	366	73385
22	Hawks	381	345	69382
23	Mets	367	365	69972
24	Rays	366	348	67993
25	Rangers	333	320	61868
26	Phillies	331	357	65147
27	Buccaneers	330	303	60598

Graph – Team Points:



3 -View

Team Schedule

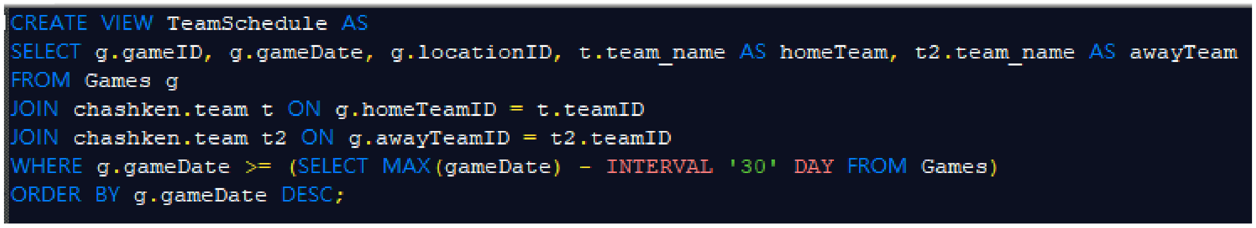
User Type:

This view will be helpful for team managers.

Description:

This view shows the complete schedule of the last 30 games for a particular team, including the opponents, game dates, and locations. It helps team managers plan and organize their team's activities and strategies.

Code:

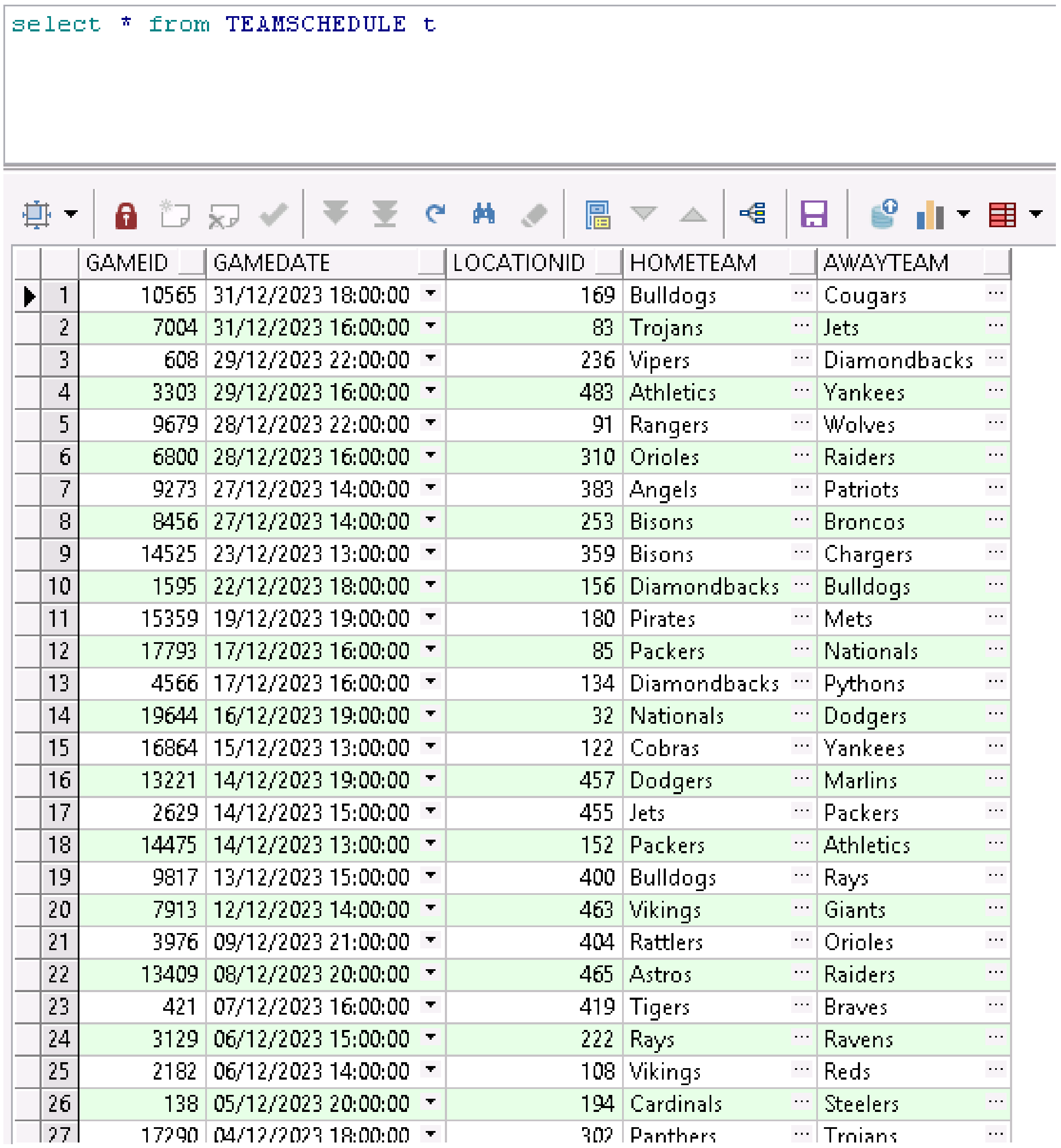


```
CREATE VIEW TeamSchedule AS
SELECT g.gameID, g.gameDate, g.locationID, t.team_name AS homeTeam, t2.team_name AS awayTeam
FROM Games g
JOIN chashken.team t ON g.homeTeamID = t.teamID
JOIN chashken.team t2 ON g.awayTeamID = t2.teamID
WHERE g.gameDate >= (SELECT MAX(gameDate) - INTERVAL '30' DAY FROM Games)
ORDER BY g.gameDate DESC;
```

PLSQL:

```
CREATE VIEW TeamSchedule AS
SELECT g.gameID, g.gameDate, g.locationID, t.team_name AS homeTeam, t2.team_name AS awayTeam
FROM Games g
JOIN chashken.team t ON g.homeTeamID = t.teamID
JOIN chashken.team t2 ON g.awayTeamID = t2.teamID
WHERE g.gameDate >= (SELECT MAX(gameDate) - INTERVAL '30' DAY FROM Games)
ORDER BY g.gameDate DESC;
```

<u>Result:</u>



```
select * from TEAMSCHEDULE t
```

	GAMEID	GAMEDATE	LOCATIONID	HOMETEAM	AWAYTEAM
1	10565	31/12/2023 18:00:00	169	Bulldogs	Cougars
2	7004	31/12/2023 16:00:00	83	Trojans	Jets
3	608	29/12/2023 22:00:00	236	Vipers	Diamondbacks
4	3303	29/12/2023 16:00:00	483	Athletics	Yankees
5	9679	28/12/2023 22:00:00	91	Rangers	Wolves
6	6800	28/12/2023 16:00:00	310	Orioles	Raiders
7	9273	27/12/2023 14:00:00	383	Angels	Patriots
8	8456	27/12/2023 14:00:00	253	Bisons	Broncos
9	14525	23/12/2023 13:00:00	359	Bisons	Chargers
10	1595	22/12/2023 18:00:00	156	Diamondbacks	Bulldogs
11	15359	19/12/2023 19:00:00	180	Pirates	Mets
12	17793	17/12/2023 16:00:00	85	Packers	Nationals
13	4566	17/12/2023 16:00:00	134	Diamondbacks	Pythons
14	19644	16/12/2023 19:00:00	32	Nationals	Dodgers
15	16864	15/12/2023 13:00:00	122	Cobras	Yankees
16	13221	14/12/2023 19:00:00	457	Dodgers	Marlins
17	2629	14/12/2023 15:00:00	455	Jets	Packers
18	14475	14/12/2023 13:00:00	152	Packers	Athletics
19	9817	13/12/2023 15:00:00	400	Bulldogs	Rays
20	7913	12/12/2023 14:00:00	463	Vikings	Giants
21	3976	09/12/2023 21:00:00	404	Rattlers	Orioles
22	13409	08/12/2023 20:00:00	465	Astros	Raiders
23	421	07/12/2023 16:00:00	419	Tigers	Braves
24	3129	06/12/2023 15:00:00	222	Rays	Ravens
25	2182	06/12/2023 14:00:00	108	Vikings	Reds
26	138	05/12/2023 20:00:00	194	Cardinals	Steelers
27	17290	04/12/2023 18:00:00	302	Panthers	Trojans

4 -View

TeamStatsSummary

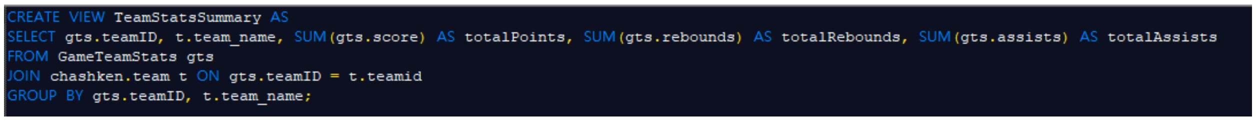
User Type:

This view will be helpful for team managers.

Description:

This view provides a summary of player statistics for each team. It includes the team ID, team name, total points scored, total rebounds, and total assists. Managers can use this view to track the overall performance of their team's players and identify the top contributors in various statistical categories

Code:

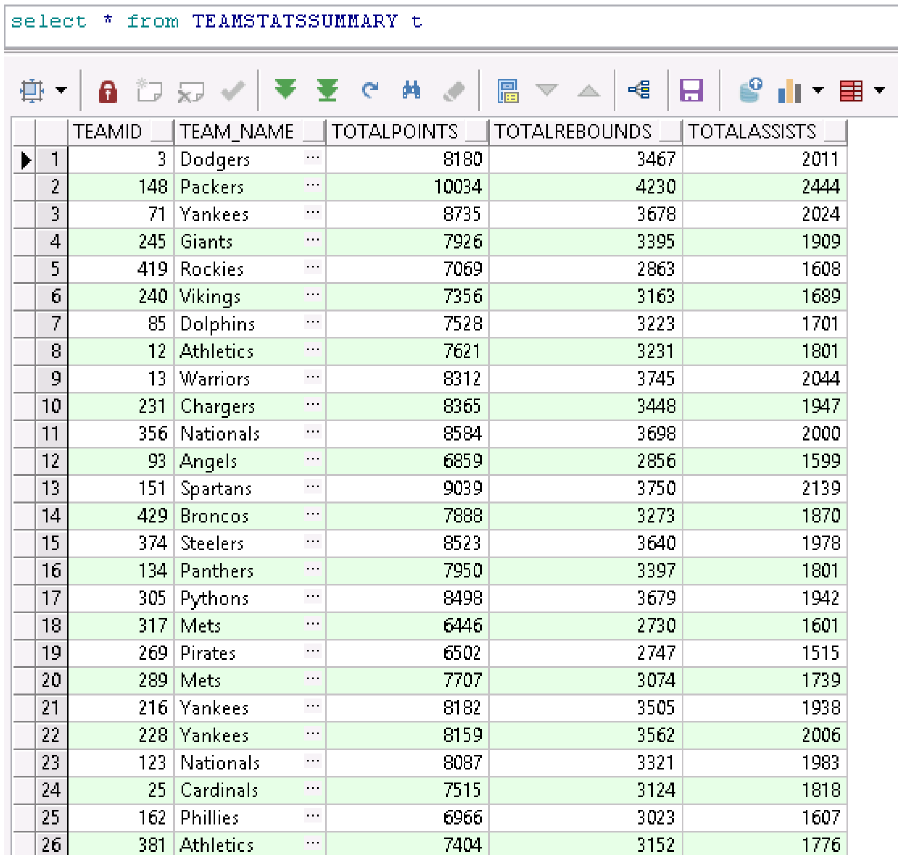


```
CREATE VIEW TeamStatsSummary AS
SELECT gts.teamID, t.team_name, SUM(gts.score) AS totalPoints, SUM(gts.rebounds) AS totalRebounds, SUM(gts.assists) AS totalAssists
FROM GameTeamStats gts
JOIN chashken.team t ON gts.teamID = t.teamid
GROUP BY gts.teamID, t.team_name;
```

PLSQL:

```
CREATE VIEW TeamStatsSummary AS
SELECT gts.teamID, t.team_name, SUM(gts.score) AS totalPoints, SUM(gts.rebounds) AS totalRebounds, SUM(gts.assists) AS totalAssists
FROM GameTeamStats gts
JOIN chashken.team t ON gts.teamID = t.teamid
GROUP BY gts.teamID, t.team_name;
```

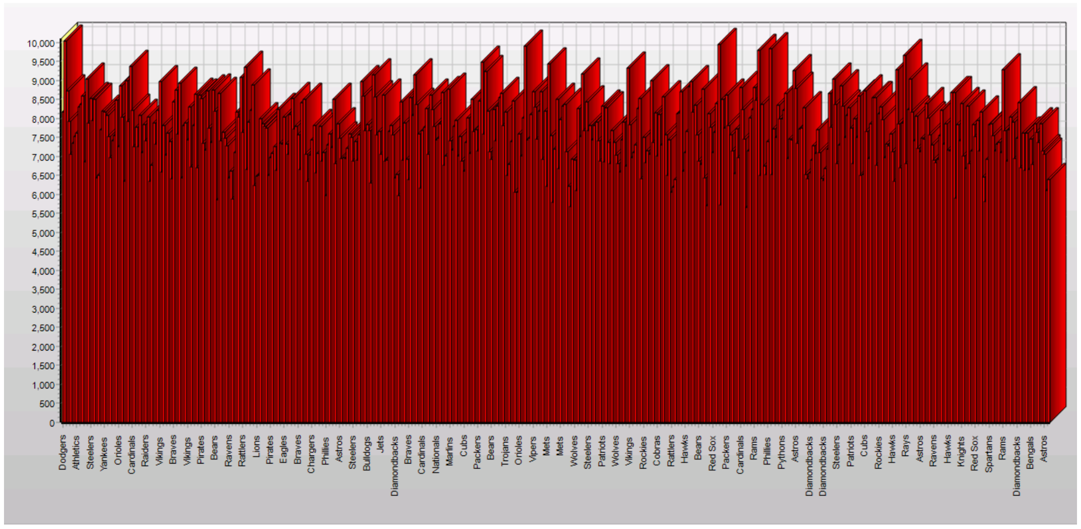
Result:



```
select * from TEAMSTATSSUMMARY t
```

	TEAMID	TEAM_NAME	TOTALPOINTS	TOTALREBOUNDS	TOTALASSISTS
1	3	Dodgers	8180	3467	2011
2	148	Packers	10034	4230	2444
3	71	Yankees	8735	3678	2024
4	245	Giants	7926	3395	1909
5	419	Rockies	7069	2863	1608
6	240	Vikings	7356	3163	1689
7	85	Dolphins	7528	3223	1701
8	12	Athletics	7621	3231	1801
9	13	Warriors	8312	3745	2044
10	231	Chargers	8365	3448	1947
11	356	Nationals	8584	3698	2000
12	93	Angels	6859	2856	1599
13	151	Spartans	9039	3750	2139
14	429	Broncos	7888	3273	1870
15	374	Steelers	8523	3640	1978
16	134	Panthers	7950	3397	1801
17	305	Pythons	8498	3679	1942
18	317	Mets	6446	2730	1601
19	269	Pirates	6502	2747	1515
20	289	Mets	7707	3074	1739
21	216	Yankees	8182	3505	1938
22	228	Yankees	8159	3562	2006
23	123	Nationals	8087	3321	1983
24	25	Cardinals	7515	3124	1818
25	162	Phillies	6966	3023	1607
26	381	Athletics	7404	3152	1776

Graph – points of team:



Procedures

1 -Procedure

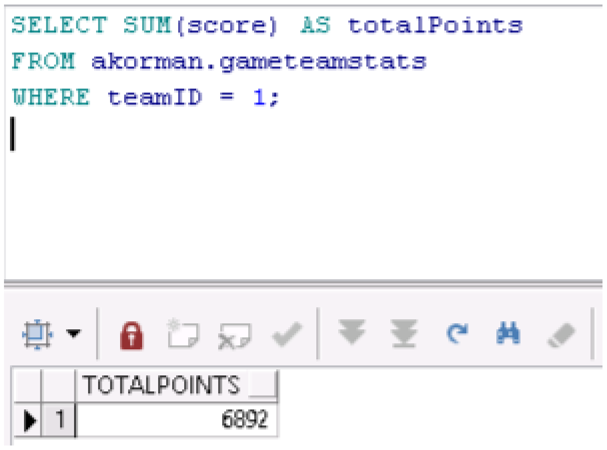
Calculate Team Points

```
CREATE OR REPLACE PROCEDURE CalculateTeamPoints(teamID IN INT)
AS
BEGIN
  UPDATE chashken.team t
  SET t.totalPoints = (SELECT SUM(gts.score)
                       FROM GameTeamStats gts
                       WHERE gts.teamID = t.teamID)
  WHERE t.teamID = teamID;
  COMMIT;
END;
```

Motivation

This procedure calculates the total points scored by a specific team and updates the totalPoints column in the chashken.team table. By storing the calculated total points in the table, it eliminates the need to perform the calculation repeatedly in queries, leading to improved runtime performance.

Before



```
SELECT SUM(score) AS totalPoints
FROM akorman.gameteamstats
WHERE teamID = 1;
```

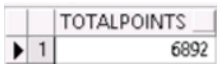
	TOTALPOINTS
1	6892

Procedure call

```
BEGIN
  -- Call the procedure with teamID = 1
  CalculateTeamPoints(1);
END;
```

After

```
SELECT totalPoints
FROM chashken.team
WHERE teamID = 1;
```



	TOTALPOINTS
1	6892

2 -Procedure

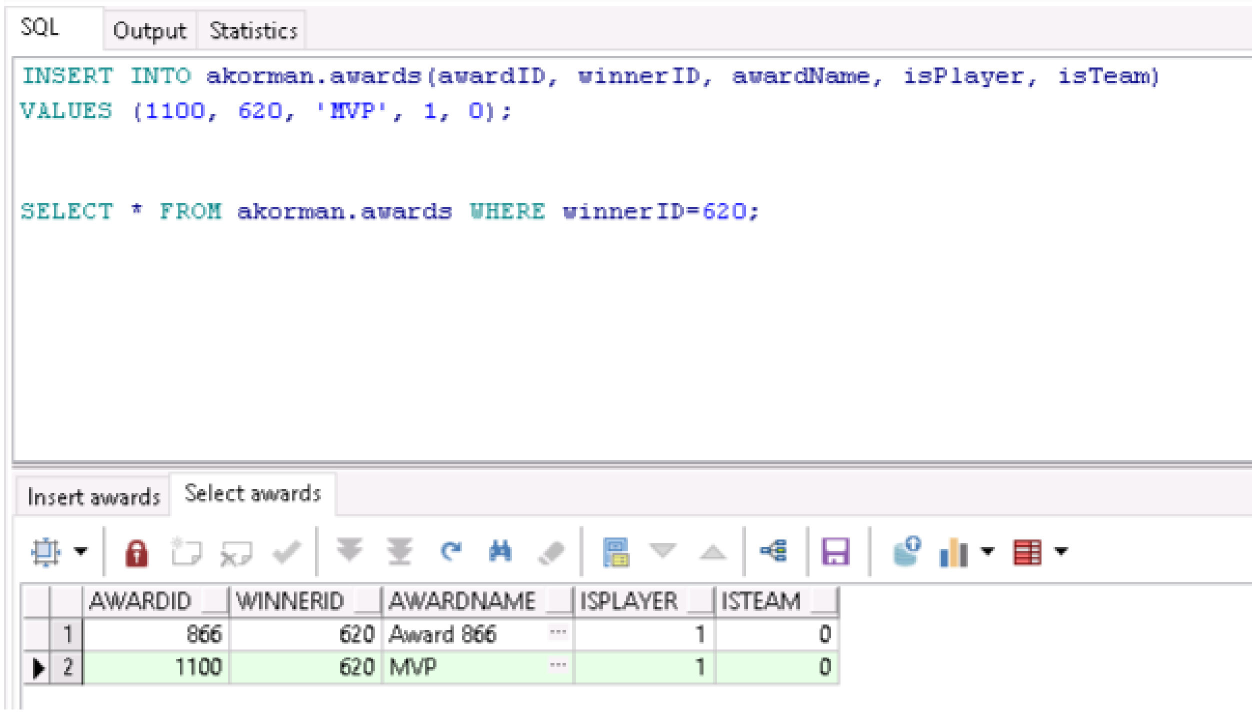
Assign Award to Player

```
CREATE OR REPLACE PROCEDURE AssignAwardToPlayer(playerID IN INT, awardName IN VARCHAR2)
AS
BEGIN
  INSERT INTO Awards (awardID, winnerID, awardName, isPlayer, isTeam)
  VALUES (award_sequence.NEXTVAL, playerID, awardName, 1, 0);
  COMMIT;
END;
```

Motivation

This procedure assigns an award to a specific player by inserting a new row into the Awards table. It simplifies the process of assigning awards to players and ensures proper data recording. By using a sequence to generate the award ID, it provides uniqueness to each award entry.

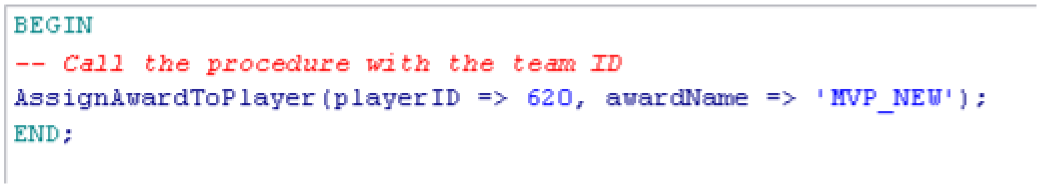
Before



```
INSERT INTO akorman.awards(awardID, winnerID, awardName, isPlayer, isTeam)
VALUES (1100, 620, 'MVP', 1, 0);


SELECT * FROM akorman.awards WHERE winnerID=620;
```

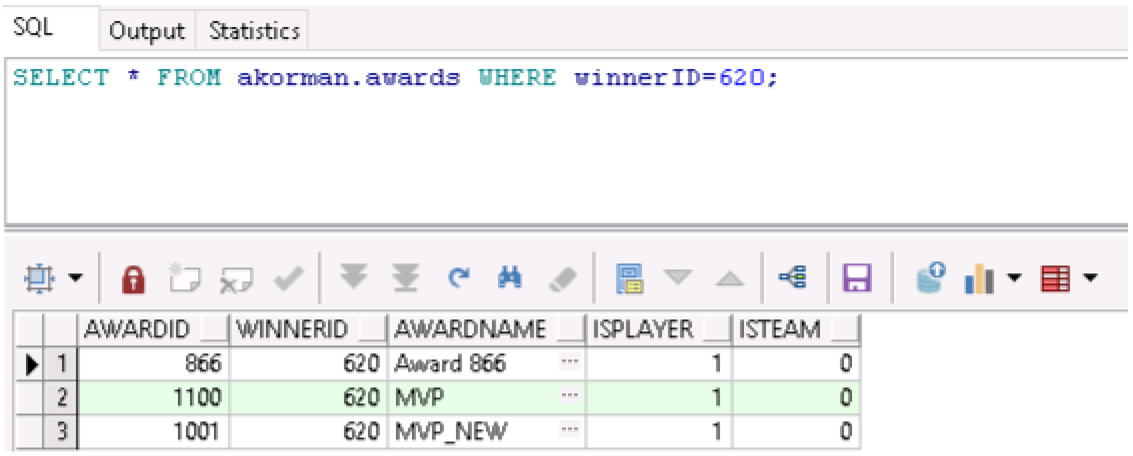
	AWARDID	WINNERID	AWARDNAME	ISPLAYER	ISTEAM
1	866	620	Award 866	1	0
2	1100	620	MVP	1	0

Procedure call



```
BEGIN
-- Call the procedure with the team ID
AssignAwardToPlayer(playerID => 620, awardName => 'MVP_NEW');
END;
```

After



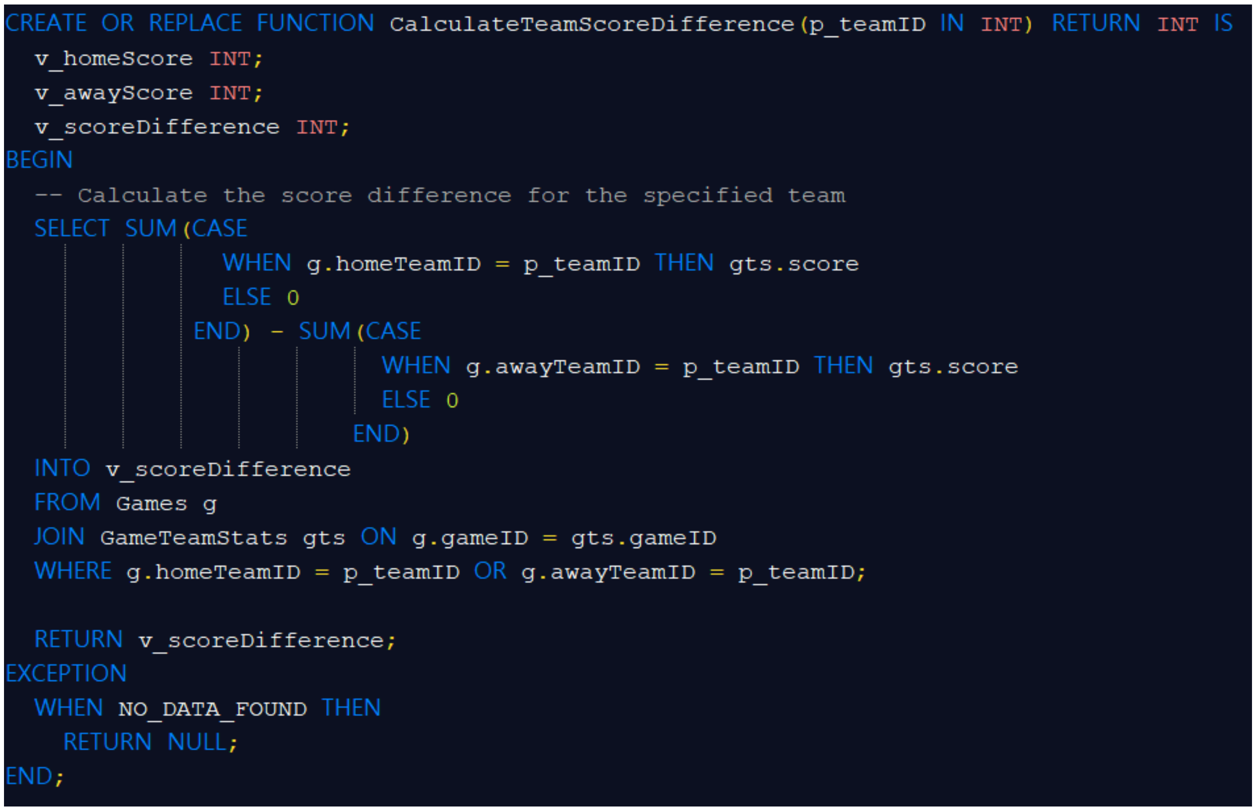
```
SELECT * FROM akorman.awards WHERE winnerID=620;
```

	AWARDID	WINNERID	AWARDNAME	ISPLAYER	ISTEAM
1	866	620	Award 866	1	0
2	1100	620	MVP	1	0
3	1001	620	MVP_NEW	1	0

Functions

1 -Function

Calculate Team Score Difference



```
CREATE OR REPLACE FUNCTION CalculateTeamScoreDifference(p_teamID IN INT) RETURN INT IS
  v_homeScore INT;
  v_awayScore INT;
  v_scoreDifference INT;
BEGIN
  -- Calculate the score difference for the specified team
  SELECT SUM(CASE
             WHEN g.homeTeamID = p_teamID THEN gts.score
             ELSE 0
           END) - SUM(CASE
                        WHEN g.awayTeamID = p_teamID THEN gts.score
                        ELSE 0
                      END)
  INTO v_scoreDifference
  FROM Games g
  JOIN GameTeamStats gts ON g.gameID = gts.gameID
  WHERE g.homeTeamID = p_teamID OR g.awayTeamID = p_teamID;

  RETURN v_scoreDifference;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    RETURN NULL;
END;
```

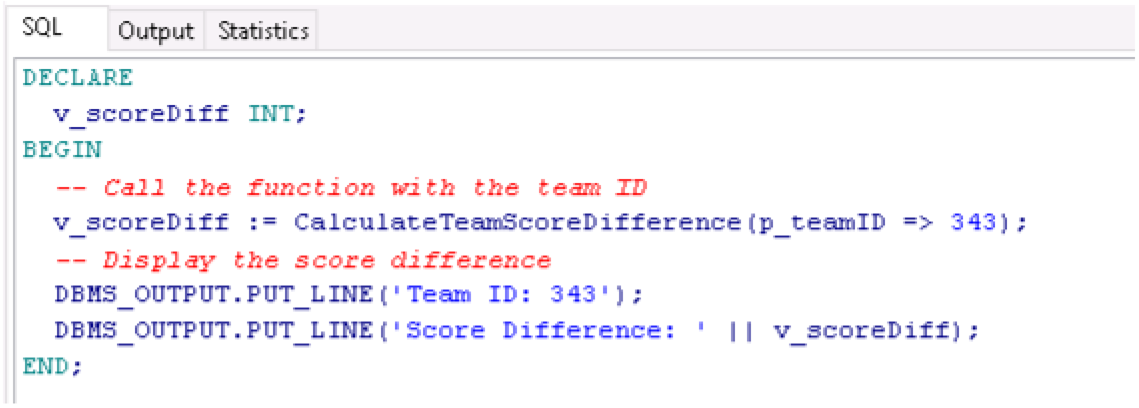
Motivation

The motivation for this function remains the same as the previous procedure. It calculates the score difference for a specific team in their games, allowing team managers to analyze their team's performance more easily.

Before

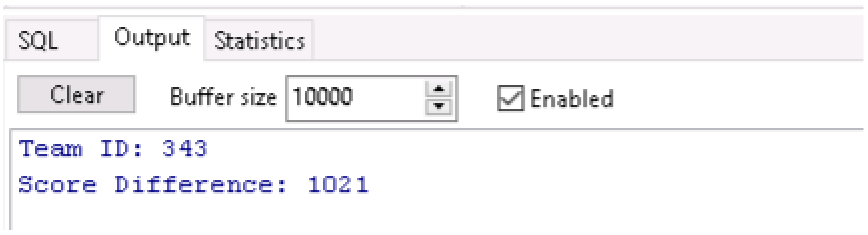
Before using this function, you would need to manually calculate the sum of the home team scores and subtract the sum of the away team scores for the desired team to determine the score difference.

Function call



```
DECLARE
  v_scoreDiff INT;
BEGIN
  -- Call the function with the team ID
  v_scoreDiff := CalculateTeamScoreDifference(p_teamID => 343);
  -- Display the score difference
  DBMS_OUTPUT.PUT_LINE('Team ID: 343');
  DBMS_OUTPUT.PUT_LINE('Score Difference: ' || v_scoreDiff);
END;
```

After

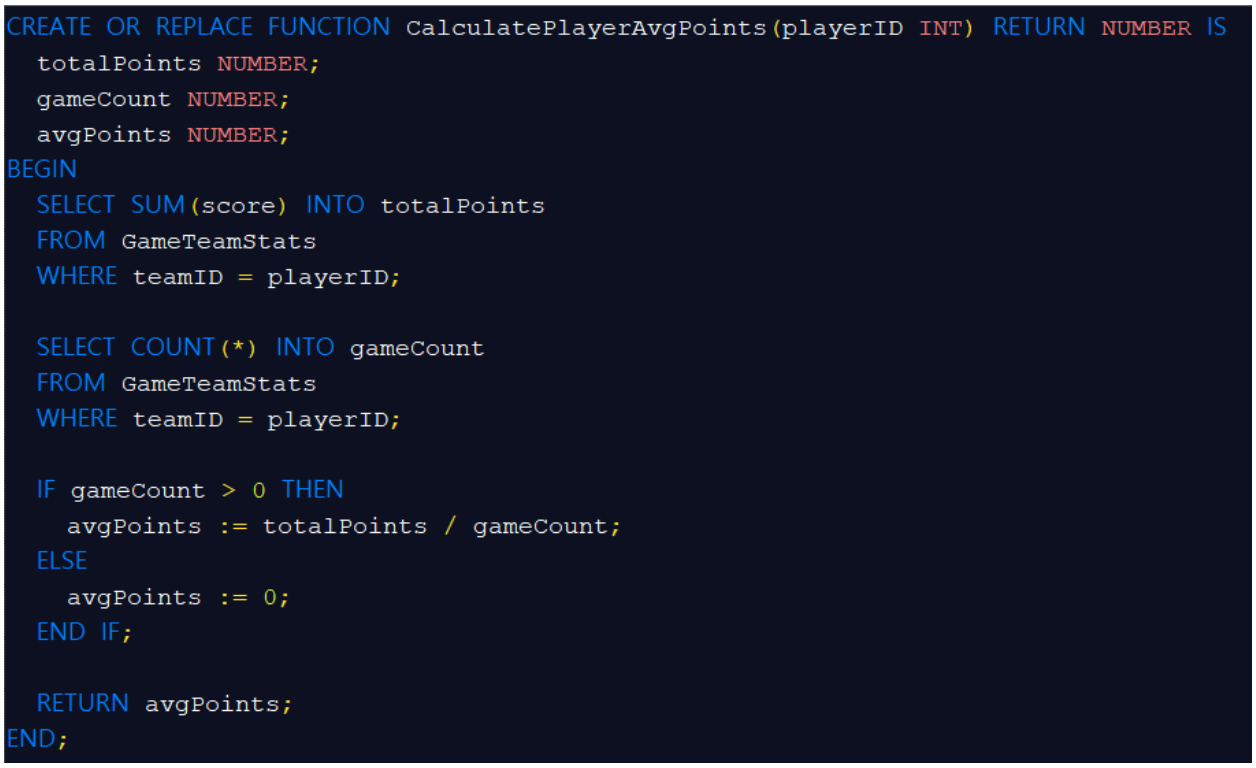


Buffer size 10000

```
Team ID: 343
Score Difference: 1021
```

2 -Function

Calculate Player Average Points



```
CREATE OR REPLACE FUNCTION CalculatePlayerAvgPoints(playerID INT) RETURN NUMBER IS
  totalPoints NUMBER;
  gameCount NUMBER;
  avgPoints NUMBER;
BEGIN
  SELECT SUM(score) INTO totalPoints
  FROM GameTeamStats
  WHERE teamID = playerID;

  SELECT COUNT(*) INTO gameCount
  FROM GameTeamStats
  WHERE teamID = playerID;

  IF gameCount > 0 THEN
    avgPoints := totalPoints / gameCount;
  ELSE
    avgPoints := 0;
  END IF;

  RETURN avgPoints;
END;
```

Motivation

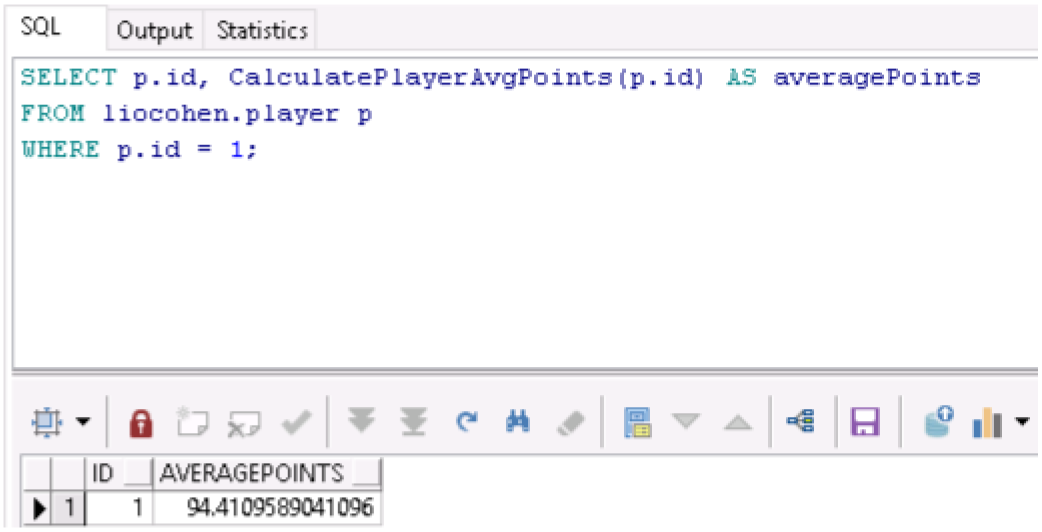
The motivation for this function is to calculate the average points scored by a player based on their performance in the games. It allows team managers and coaches to assess the scoring capabilities of individual players and make informed decisions regarding game strategies and player roles.

Before

Before using the function, you would need to manually calculate the average points scored by a player by executing multiple SQL queries to retrieve the total points and the number of games played by the player. This process can be time-consuming and error prone.

Function call

1. Retrieve the average points for a specific player:

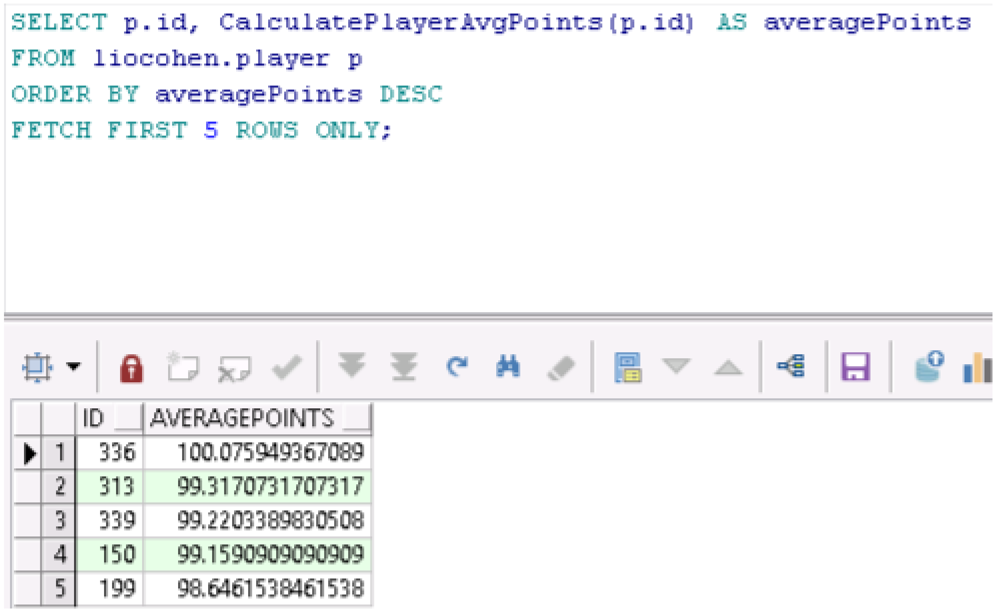


SQL Output Statistics

```
SELECT p.id, CalculatePlayerAvgPoints(p.id) AS averagePoints
FROM liocohen.player p
WHERE p.id = 1;
```

	ID	AVERAGEPOINTS
1	1	94.4109589041096

2. Find the top 5 players with the highest average points:



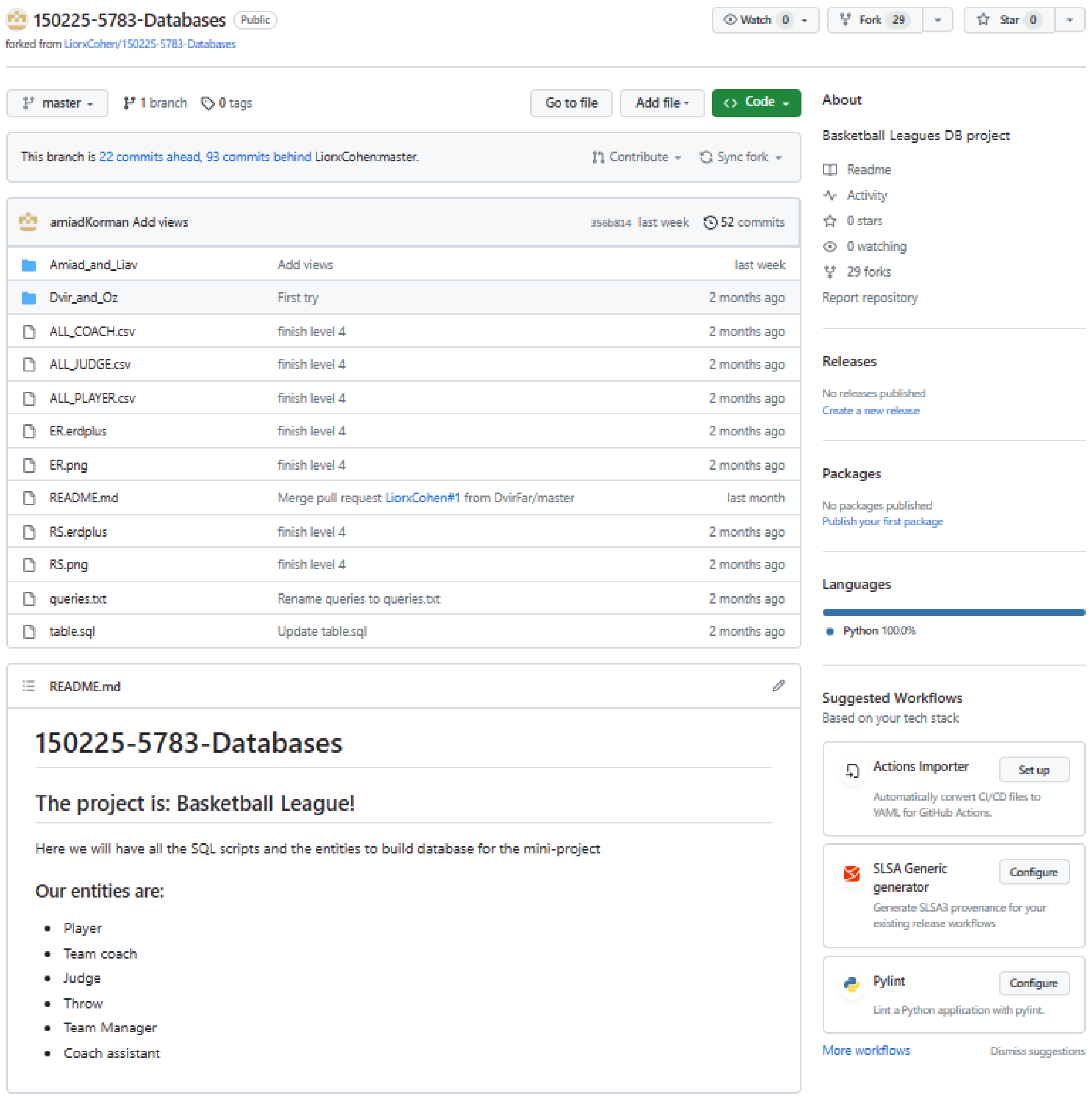
```
SELECT p.id, CalculatePlayerAvgPoints(p.id) AS averagePoints
FROM liocohen.player p
ORDER BY averagePoints DESC
FETCH FIRST 5 ROWS ONLY;
```

	ID	AVERAGEPOINTS
1	336	100.075949367089
2	313	99.3170731707317
3	339	99.2203389830508
4	150	99.1590909090909
5	199	98.6461538461538

After

After implementing the function, you can simply call it with the player's ID, and it will calculate the average points in a more efficient and convenient way. The function encapsulates the necessary calculations and returns the average points directly, saving time and effort in manual calculations.

Git



150225-5783-Databases Public

forked from LiorxCohen/150225-5783-Databases

Watch 0 · Fork 29 · Star 0

master · 1 branch · 0 tags · Go to file · Add file · Code

This branch is 22 commits ahead, 93 commits behind LiorxCohen:master. Contribute · Sync fork

amiadKorman Add views 356b814 last week 52 commits

Amiad_and_Liav	Add views	last week
Dvir_and_Oz	First try	2 months ago
ALL_COACH.csv	finish level 4	2 months ago
ALL_JUDGE.csv	finish level 4	2 months ago
ALL_PLAYER.csv	finish level 4	2 months ago
ER.erdplus	finish level 4	2 months ago
ER.png	finish level 4	2 months ago
README.md	Merge pull request LiorxCohen#1 from DvirFar/master	last month
RS.erdplus	finish level 4	2 months ago
RS.png	finish level 4	2 months ago
queries.txt	Rename queries to queries.txt	2 months ago
table.sql	Update table.sql	2 months ago

README.md

150225-5783-Databases

The project is: Basketball League!

Here we will have all the SQL scripts and the entities to build database for the mini-project

Our entities are:

- Player
- Team coach
- Judge
- Throw
- Team Manager
- Coach assistant

About

Basketball Leagues DB project

Readme · Activity · 0 stars · 0 watching · 29 forks · Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

Python 100.0%

Suggested Workflows
Based on your tech stack

Actions Importer — Set up — Automatically convert CI/CD files to YAML for GitHub Actions.

SLSA Generic generator — Configure — Generate SLSA3 provenance for your existing release workflows

Pylint — Configure — Lint a Python application with pylint.

More workflows · Dismiss suggestions

Link: https://github.com/amiadKorman/150225-5783-Databases.git

References:

- https://www.oracletutorial.com/plsql-tutorial/
- https://mockaroo.com/
- https://www.geeksforgeeks.org/sql-indexes/
- https://stackoverflow.com/questions/2955459/what-is-an-index-in-sql
- https://medium.com/@kishlay.kumar/sql-indexing-why-is-it-important-836fe80837e6
- https://www.geeksforgeeks.org/sql-views/
- https://www.w3schools.com/SQL/sql_stored_procedures.asp
- https://learn.microsoft.com/en-us/sql/t-sql/statements/create-function-transact-sql?view=sql-server-ver16

Additional Reference Materials:

- Recordings and materials provided by the lecturer.
- https://docs.google.com/spreadsheets/d/1M6zAZbKwIK7s4U-mniCKvBqlxmh60oTyiqIQdotG7WA/edit#gid=0.