Basketball Database Term Project

Yusen Zhou

Boston University

CS669 Database Design and Implementation for Business

August 11, 2023

Contents

Basketball Database Term Project Iteration 1

Project Direction Overview

I would like to develop an NBA data app which will provide team or player performance historical data to the users, like ESPN. I named this app "BasketballStats". I'm an NBA fan, and there are tons of data in this league. Fans use statistics to judge whether players are playing well and vote for All-Stars. They can even predict victories using past stats, which is especially important in these years when sports betting is legalized. While ESPN also provides sports stats, it would take a lot of time for someone to try to extract stats for a specific game, especially ones from a long time ago. Not to mention that if fans want to apply filters to the data, such as find players who average more than 30 points this season. A database storing all NBA statistics will help users extract any needed data efficiently, usually in less than a second. Immediately after each NBA game, a new record is inserted into the database to keep it up to date.

Use Cases and Fields

One important usage of the database is when a person wants to see the complete list of game played between the supported team and the opponents.

Game History Use Case:

1. People visit BasketballStats' website or application.
2. The app asks them to select the team they support or are interested in.
3. The app asks them to select the opponent team (if null, then returns all past games).
4. The app returns historical data including total games, win and losses, scores, date, and etc..

Step #5 highlights that the database will store relevant information about an NBA games. Significant fields are detailed below.

Field	What it Stores	Why it's Needed
GameDate	This is the date of the game played.	This is useful so that the fan knows what day the game was played, and so that they can search for the exact game or period they desire.
Teams	These are the names of the home and away teams.	This is useful so the fan knows that the team he/she is interested in is playing home/away.
Arena	This is the arena where the game was played.	This is necessary so that the person knows what arena is holding the match, and ticket sales can be linked.

Scores	This is the score result of the game.	This is useful so that the fan knows which team won the game and the score difference.
BestPlayer	This is the name of the highest scoring player on the winning team	This is useful so the person knows who the best player in that game.
Description	This tells whether it's the regular season or playoff.	This is necessary because playoff games are much more important than regular season games.

Another important usage of the database is when a fan wants to see a player's profile. Player Profile Look Up Use Case:

1. The website/app asks the user to input a player name.
2. The website/app returns the player profile.

Field	What it Stores	Why it's Needed
PlayerName	This tells the player's name.	This is useful so that the person knows which player they are investigating.
CurrentTeam	This tells the team this player is currently playing for.	This is useful so that the person knows what team the player is playing for.
Height&Weight	This tells the size of the player.	This is useful when the person wants to compare players' sizes.
Age	This tells the age of the player.	This is useful when the fan wants to compare players' ages.
DraftInfo	This tells the draft pick of the player.	This is useful when the person wants to see if a player is performing as expected by the team scouts.

Another important usage is when a person decides to lookup the full box score of a specific game. Full Box Score Use Case:

1. The app asks the user which teams to view (same as in the first use case).
2. The person clicks on the "Full Box Score" button next to a specific game.
3. The app returns complete player's and team's stats of the game.

Field	What it Stores	Why it's Needed
Team	This tells the team of this player.	This is necessary so that the table is separated into two parts, one for the home team and the other for the away team.
Player	This tells the players' names.	This is useful so that the person knows which player they are investigating.
Minutes	This tells how many minutes the player has played.	This is useful so that the fan knows how long this player has played and how much

		he contributed by looking at other stats together.
Points	This tells how many points the player has scored.	This is useful so that the fan knows who has the highest score.
Assist	This tells how many times the player has assisted.	This is useful so that the fan knows who has the highest amount of assist.
Rebound	This tells how many rebounds the player has grabbed.	This is useful so that the fan knows who grabbed the most rebound.
(MoreStats)	(e.g., steals, blocks, 3pt%, personal foul, +/-, etc.)	(Similar to above)

Summary and Reflection

My database is a website/mobile app named BasketballStats which records all the match result, team historical data, and players stats. Normally, when a fan tries to get a specific old game stat on ESPN.com, it would take the person plenty of time. BasketballStats aims to provide an efficient and customizable way of extracting NBA statistics for fans. The database must support a person entering, searching, and applying filters across all historical statistics. Three use cases are also explained above. Any additional feedback is appreciated.

Basketball Database Term Project Iteration 2

BasketballStats Structural Rules

Game History Use Case

1. A user visits the BasketballStats database.
2. The website asks them to select the team they support or are interested in.
3. The website asks them to select the opponent team (if null, then returns all past games).
4. The website returns all the historical records that contain these two teams. Each record includes game date, teams' name, arena, final scores, winner's and loser's player of game (POG) and note.

From this use case, I see many entities – Game, Team, Player, Arena, and Box Score. Structural database rules are as below:

a. Each game must have two teams; each team may have many games.
(If a new team joins the NBA, before the first game, it doesn't have any previous games. So, it's a 'may')
b. Each game must have two POGs; each player may have many times POGs.
c. Each game must have only one arena; each arena might host many games.
d. Each game must have only one box score; each box score must correspond to only one game.

Player Profile Use Case

1. The website asks the user to input a player name.
2. The website returns the player profile, which includes attribute like player name, current team, height, weight, age, draft pick, and season average stats.

From this use case, included entities are Player and Team. Structural database rules are as below:

a. Each team must have many players; each player may be in a team.
(free agents don't have a team)

Box Score Use Case

1. The person clicks on the "Full Box Score" button next to a specific game.
2. The website returns complete player's and team's stats of the game.

From this use case, included entities are Player, Team, and Box Score. Structural database rules are as below:

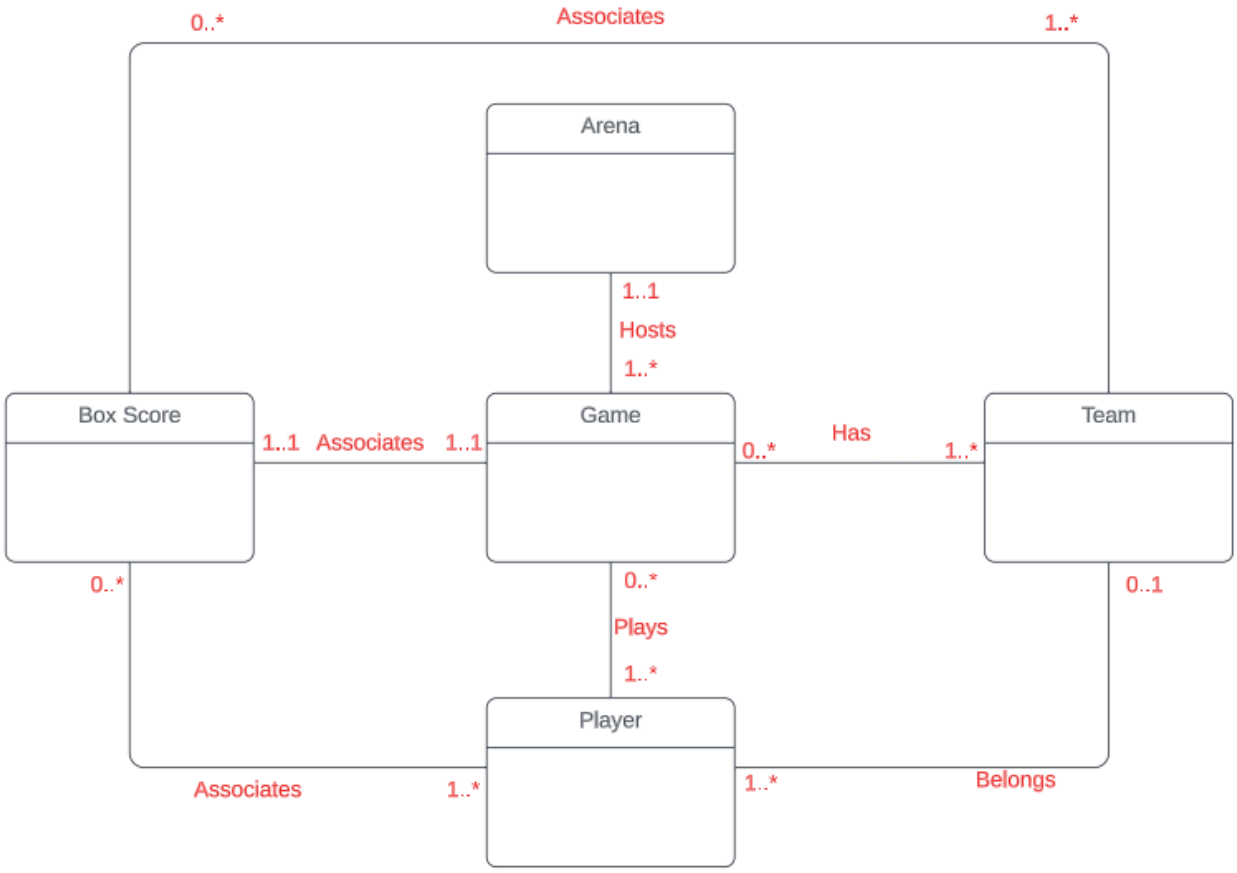
a. Each box score must include many players' statistic; each player may have many box score history.
b. Each box score must include two teams; each team may have many box score history.

Initial BasketballStats ERD

Structural database rules relist:

a. Each game must have two teams; each team may have many games.
b. Each game must have two POGs; each player may have many times POGs.
c. Each game must have only one arena; each arena might host many games.
d. Each game must have only one box score; each box score must correspond to only one game.
e. Each team must have many players; each player may be in a team.
f. Each box score must include many players' statistic; each player may have many box score history.
g. Each box score must include two teams; each team may have many box score history.

Here is the ERD I came up with for these rules using UML Class Diagram Notation:



Basketball Database Term Project Iteration 3

Initial BasketballStats Structural Rules (From Iteration 2)

Structural database rules from iteration 2 re-list:

a. Each game is associated with two teams; each team may be associated with many games.
b. Each game has two POGs (Player of the Game); each player may be many times POGs.
c. Each game is hosted in one arena; each arena may host many games.
d. Each game has one box score (table); each box score is associated to one game.
e. Each team has one to many players; each player may be in one team.
f. Each box score includes many players' statistic; each player may have many box score history.
   Each box score includes two teams; each team may have many box score history.
   (removed, because Team links to Game, and Game links to Box Score)

Adding Specialization-Generalization to BasketballStats

Game Schedule & Result Use Case

1. A user visits the BasketballStats database.
2. The website asks them to select the team they support or are interested in.
3. The website asks them to select the opponent team (if null, then returns all past & future game information of the selected team).
4. The website returns all the historical records that contain these two teams. Each record includes game date, teams' name, arena, final scores, winner's and loser's player of game (POG) and note.

In an effort to increase views/usage and serve a wider range of basketball fans, BasketballStats has decided to offer college basketball and other pro-basketball league statistics as well. Therefore, there is an additional step, *select the league*, before the original #2 step.

A new structural database rule to support the change to the use case as follows.

g. A basketball game is a NCAA game, a G-League game, an NBA game or none of these.

My database currently collects stats of three leagues – NCAA, G-League, NBA. But these are not the only leagues available in the real-life case, there are many other international associations like FIBA, Euro League, Chinese Basketball Association. So, I made this relationship partially complete to allow flexibility for games in other leagues. That is why I used phrase "or none of these". Verbiage like "several of these" is not in the sentence since a basketball game cannot be officially played between two teams in different leagues.

Box Score Use Case

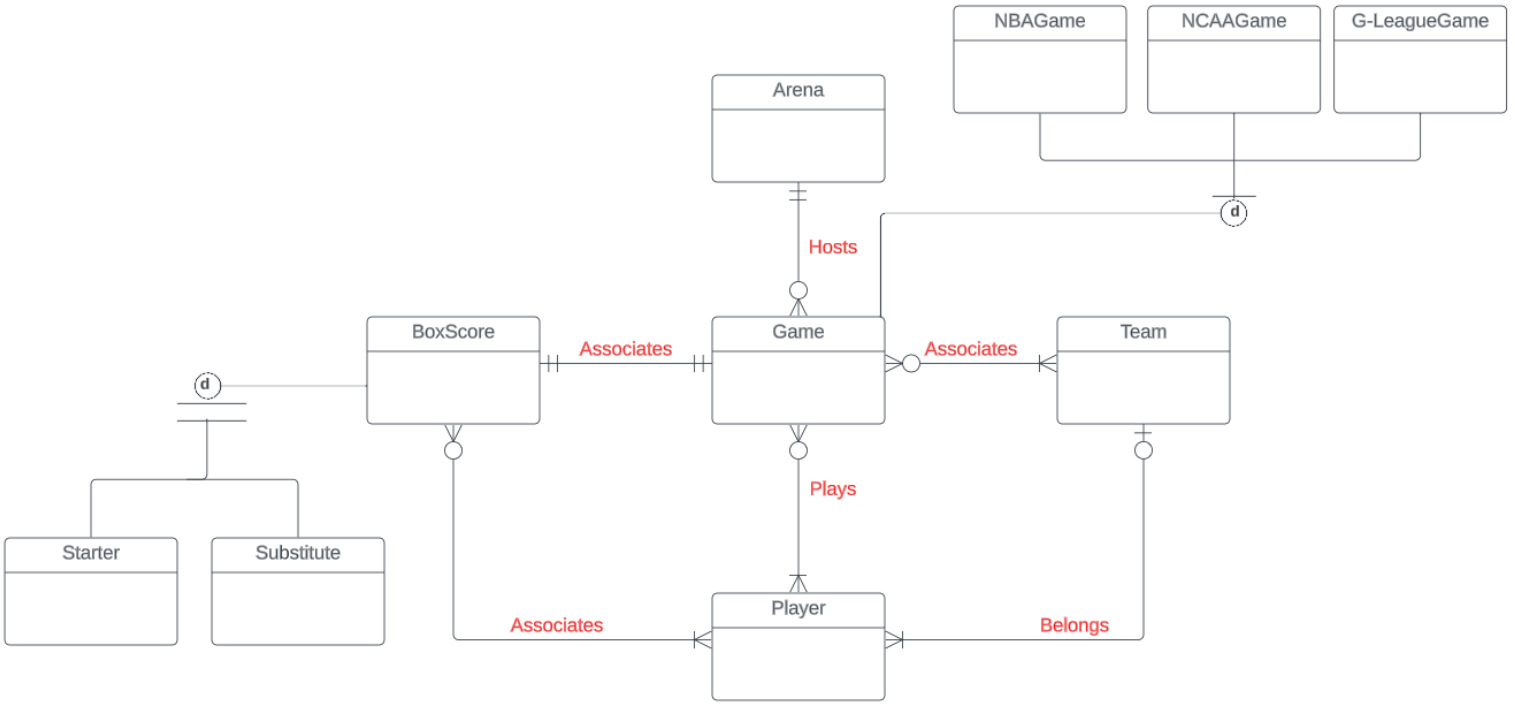
1. The person clicks on the “Full Box Score” button next to a specific game.
2. The website returns complete player’s and team’s stats of the game.

In this use case, a player can be a starter or a bench player. A new structural database can added as follows.

h. A player is a starter or a bench player.

In a basketball game, a player participating in a game who does not start is automatically considered a substitute for that game. Verbiage such as “several of these” or “none of these” are not needed since the rule is totally complete and disjoint.

Two additional structural database rules are added to my ERD as follows.



BasketballStats Relationship Classification and Associative Mapping

The associative relationships in my conceptual ERD are BoxScore/Game, BoxScore/Player, Arena/Game, Game/Player, Game/Team, and Player/Team.

The BoxScore/Game relationship is 1:1.

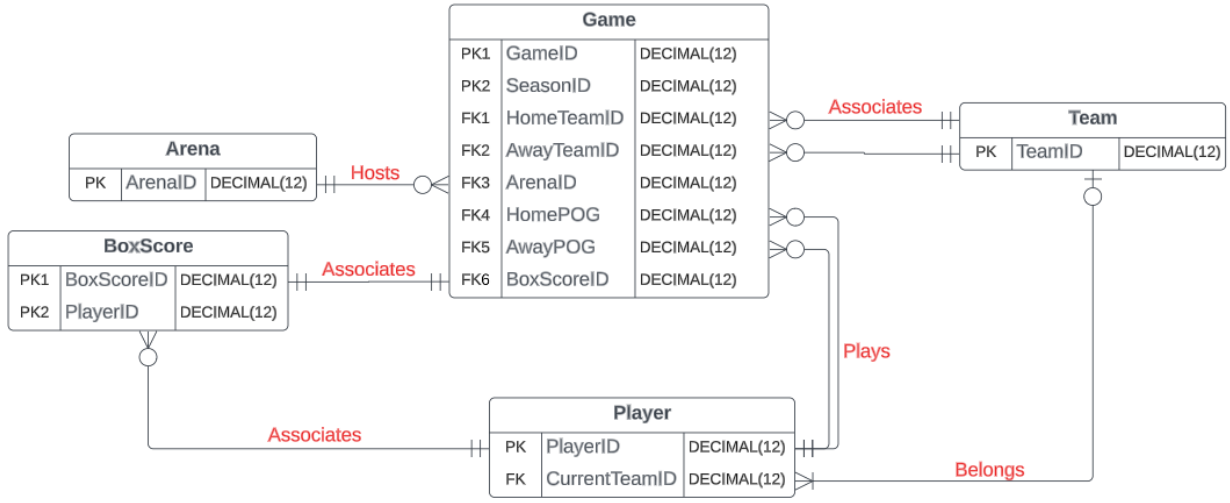
The BoxScore/Player relationship is 1:M. It was M:N, however, by building a composite key, the problem is avoided.

The Arena/Game relationship is 1:M.

The Game/Player relationship is 1:M. It was M:N since a game has two POGs, however, by setting two separate attributes and foreign keys, the problem is avoided.

The Game/Team relationship is 1:M. It was M:N since a game has two teams, however, by setting two separate attributes and foreign keys, the problem is avoided.

The Player/Team relationship is 1:M.

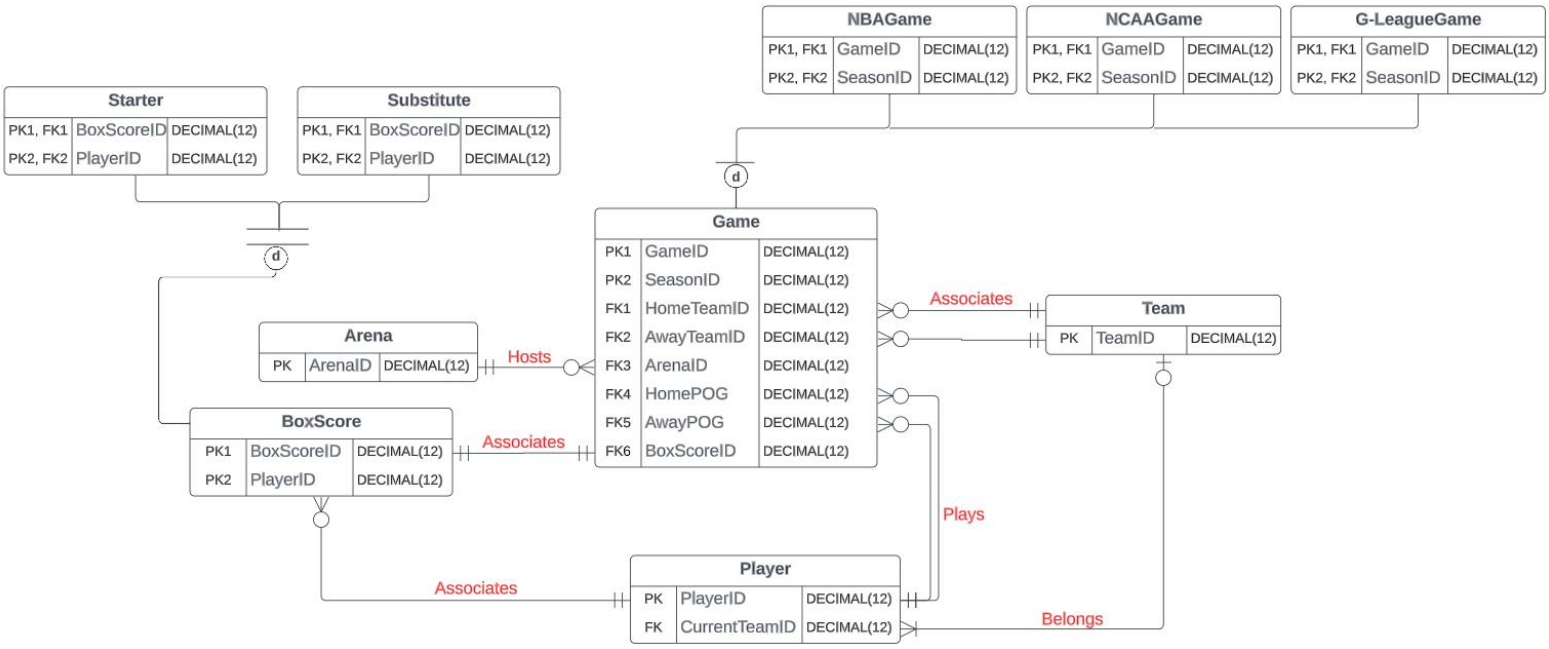


BasketballStats Specialization-Generalization Mapping

I have two specialization-generalization relationships in my conceptual ERD, one for the Game entity and one for the BoxScore entity. Here is my DBMS physical ERD with these relationships mapped into them.

The additional entities under BoxScore are Starter and Substitute, each of which have a composite primary key and foreign key of BoxScoreID plus PlayerID which reference to the composite primary key of BoxScore.

The additional entities under Game are NBAGame, NCAAGame and G-LeagueGame, each of which have a composite primary key and foreign key of GameID plus SeasonID which reference to the composite primary key of Game.



Adding Attributes to the DBMS Physical ERD

Table	Attribute	Datatype	Reasoning
Arena	Arena Name	VARCAHR(50)	Every arena has a name which acts like the identifier for the arena. The 50-character limit should be more than enough.
Arena	Address	VARCHAR(100)	Every arena has an address. This is for the address name only.
Arena	City	VARCHAR(50)	Cities are listed separately for easy retrieval
Game	NA	NA	No additional attributes are needed at this stage.
BoxScore	Minutes, Shot Made, Shot Attempted, 3PT Made, 3PT Attempted, FT Made, FT Attempted, Offensive Rebound, Defensive Rebound, Assist, Block, Steal,	DECIMAL(3)	These are basic traditional must-have player statistics. Also, these data would not exceed 2-digit whole number. I allow for up to 3-digit just in case of something extraordinary. Percentages can be derived from Made and Attempted.

	Personal Foul, Turnover, Points Score, Plus/Minus;		
Player	Birthdate	DATE	I choose birthday instead of age because doing so automatically calculates age and avoids manual updates.
Player	Status	VARCHAR(10)	Player injury report. Either be ‘Healthy’, ‘Day-To-Day’, or ‘Out’.
Player	Position	VARCHAR(15)	Can only be ‘Point Guard’, ‘Shooting Guard’, ‘Small Forward’, ‘Power Forward’, or ‘Center’. This is the position the player mainly play.
Player	Draft Info	VARCHAR(23)	A record of the player draft information. Format as ‘YEAR: Rd #, Pk ## (CIT)’. Allow only up to 23-digit, because the lowest pick is 60, which would not exceed 2-digit. Same for Round.
Player	Height/Weight	VARCHAR(16)	A record of the player body information. Format as ‘#’ ##”, ### lbs’. Allow only up to 16-digit, because if it exceeds 16 digits, it means that it is not entered according to the format.
Team	Team Name, Division Name, Conference Name	VARCHAR(12)	Each team belongs to one of the divisions: Atlantic, Central, Southeast, Northwest, Pacific, and Southwest. The first three divisions belong to the Eastern Conference, and the other three belong to the Western Conference.

Basketball Database Term Project Iteration 4

Initial BasketballStats Structural Rules (From Iteration 3)

Structural database rules from iteration 2 re-list:

a. Each game is associated with two teams; each team may be associated with many games.
b. Each game has two POGs (Player of the Game); each player may be many times POGs.
c. Each game is hosted in one arena; each arena may host many games.
d. Each game has one box score (table); each box score is associated to one game.
e. Each team has one to many players; each player may be in one team.
f. Each box score includes many players' statistic; each player may have many box score history.
   Each box score includes two teams; each team may have many box score history.
   (removed, because Team links to Game, and Game links to Box Score)
g. A basketball game is a NCAA game, a G-League game, an NBA game or none of these.
h. A player is a starter or a bench player.

Adding Attributes to the DBMS Physical ERD

Table	Attribute	Datatype	Reasoning
Arena	Arena Name	VARCAHR(50)	Every arena has a name which acts like the identifier for the arena. The 50-character limit should be more than enough.
Arena	Address	VARCHAR(100)	Every arena has an address. This is for the address name only.
Arena	City	VARCHAR(50)	Cities are listed separately for easy retrieval
Game	GameDate	DATE	Game date.
BoxScore	Minutes, Shot Made, Shot Attempted, 3PT Made, 3PT Attempted, FT Made, FT Attempted, Offensive Rebound, Defensive Rebound, Assist, Block, Steal, Personal Foul, Turnover, Points Score, Plus/Minus;	DECIMAL(3)	These are basic traditional must-have player statistics. Also, these data would not exceed 2-digit whole number. I allow for up to 3-digit just in case of something extraordinary. Shooting percentages can be derived from Made and Attempted.

BoxScore	Starter	VARCHAR(3)	A Boolean value of "yes" or "no" indicates whether the player is in the starting lineup for the game.
Player	FirstName, LastName	VARCHAR(20)	Name of the player.
Player	Birthdate	DATE	I choose birthday instead of age because doing so automatically calculates age and avoids manual updates.
Player	Status	VARCHAR(10)	Player injury report. Either be ‘Healthy’, ‘Day-To-Day’, or ‘Out’.
Player	Position	VARCHAR(15)	Can only be ‘Point Guard’, ‘Shooting Guard’, ‘Small Forward’, ‘Power Forward’, or ‘Center’. This is the position the player mainly play.
Player	Draft Info	VARCHAR(23)	A record of the player draft information. Format as ‘YEAR: Rd #, Pk ## (CIT)’. Allow only up to 23-digit, because the lowest pick is 60, which would not exceed 2-digit. Same for Round.
Player	Height/Weight	VARCHAR(16)	A record of the player body information. Format as ‘#’ ##’’, ### lbs’. Allow only up to 16-digit, because if it exceeds 16 digits, it means that it is not entered according to the format.
Team	Team Name, Division Name, Conference Name, (University Name, Affiliated NBA Team)	VARCHAR(30)	Each team belongs to different divisions and conference. The NCAA Team has an additional attribute, University Name. G-League Team has an additional attribute, Affiliated NBA Team.
Trade	Player ID, Old Team ID, New Team ID	DECIMAL(12)	After completing lab 4, I decided to add a history table to collect player's team changes.
Trade	Trade Date	DATE	The date a player is getting traded.

BasketballStats Normalization

- *Conversion to 1NF:*

I planned to use a composite primary key for the Game table, which consists of three attributes, which are GameID, SeasonID, and LeagueID. However, I noticed that BoxScoreID can also uniquely identify the Game table, since each game has one BoxScoreID and each BoxScoreID relates to one game. To adjust, I decide to use the GameID solely as the single-attribute primary key for the Game table and delete BoxScoreID. SeasonID and LeagueID remain in the table, but

they are no longer part of the primary key. Part of the BoxScore table PK is replaced by GameID as well.

- *Conversion to 2NF:*

The Box Score is the only table with a composite key, but it does not have any partial dependencies. All the nonprime attributes in Box Score table are dependent on both components of primary key. All the other tables have a single-attribute primary key. So, no changes are needed.

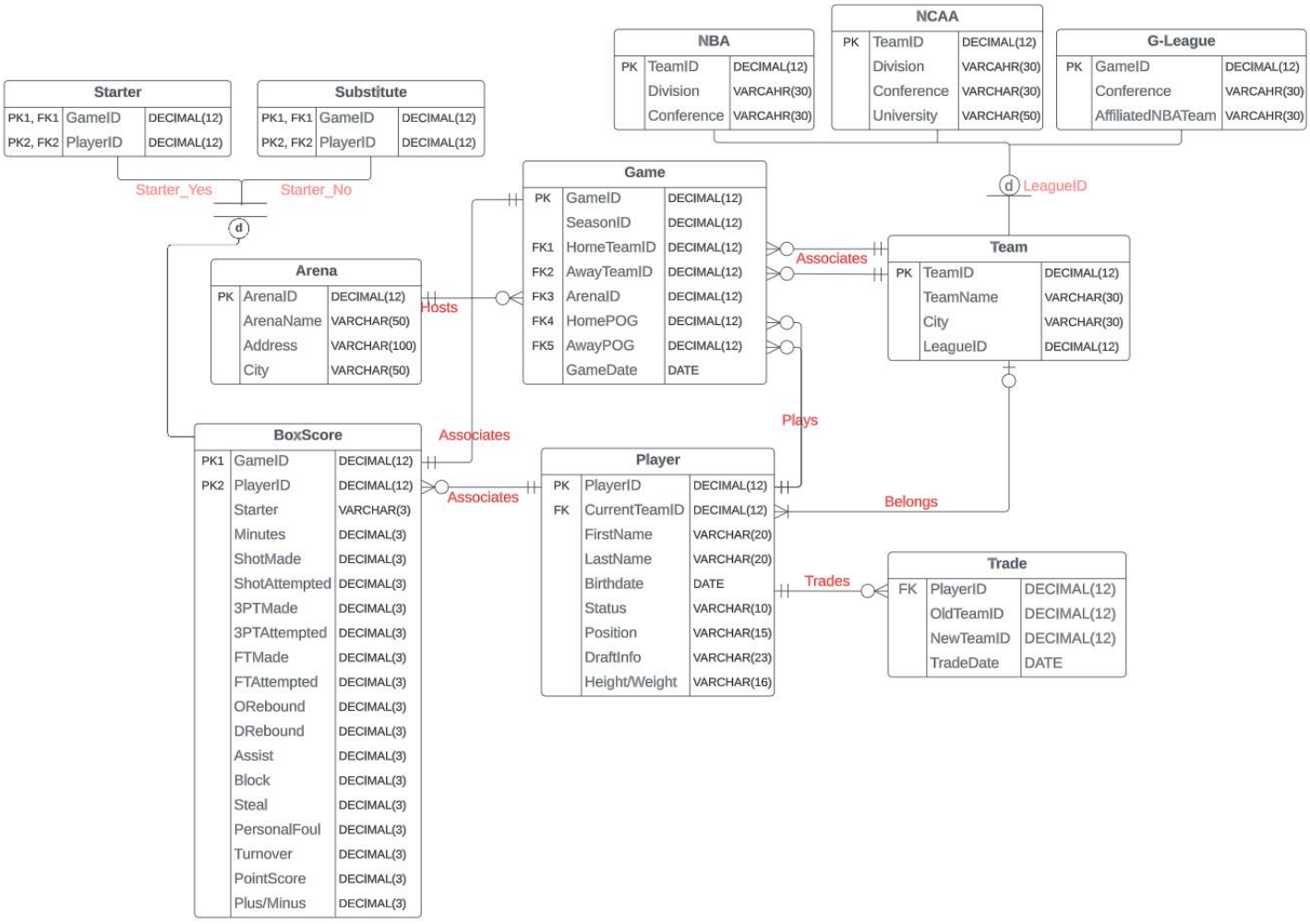
- *Conversion to 3NF:*

After checking, no functional dependency exists among nonprime attributes, which means there are no transitive dependencies. No changes are needed.

- *Conversion to BCNF:*

After checking, all the determinants in the tables are candidate keys. My ERD is in BCNF already.

Here is the final look of my ERD:



BasketballStats Create Script

```
DROP TABLE Trade;
DROP TABLE Substitute;
DROP TABLE Starter;
DROP TABLE box_score;
DROP TABLE Game;
DROP TABLE Arena;
DROP TABLE Player;
DROP TABLE G_league;
DROP TABLE NCAA;
DROP TABLE NBA;
DROP TABLE Team;

CREATE TABLE Team(
team_id   DECIMAL(12) PRIMARY KEY,
team_name VARCHAR(30) NOT NULL,
city      VARCHAR(30) NOT NULL,
league_id DECIMAL(12));

CREATE TABLE NBA(
team_id    DECIMAL(12) PRIMARY KEY,
FOREIGN KEY (team_id) REFERENCES Team(team_id),
division   VARCHAR(30) NOT NULL,
conference VARCHAR(30) NOT NULL);

CREATE TABLE NCAA(
team_id    DECIMAL(12) PRIMARY KEY,
FOREIGN KEY (team_id) REFERENCES Team(team_id),
division   VARCHAR(30) NOT NULL,
conference VARCHAR(30) NOT NULL,
university VARCHAR(50) NOT NULL);

CREATE TABLE G_League(
team_id        DECIMAL(12) PRIMARY KEY,
FOREIGN KEY (team_id) REFERENCES Team(team_id),
conference     VARCHAR(30) NOT NULL,
affiliated_nba VARCHAR(50));

CREATE TABLE Player(
player_id DECIMAL(12) PRIMARY KEY,
current_team_id DECIMAL(12),
FOREIGN KEY (current_team_id) REFERENCES team(team_id),
first_name    VARCHAR(20) NOT NULL,
last_name     VARCHAR(20) NOT NULL,
birthdate     DATE        NOT NULL,
status        VARCHAR(10) NOT NULL CHECK (status IN ('Healthy', 'Day-To-Day', 'Out')),
position      VARCHAR(15) NOT NULL CHECK (position IN('Point Guard', 'Shooting Guard', 'Small Forward', 'Power Forward', 'Center')),
draft_info    VARCHAR(23) NOT NULL,
height_weight VARCHAR(16) NOT NULL);

CREATE TABLE Arena(
arena_id   DECIMAL(12) PRIMARY KEY,
arena_name VARCHAR(50) NOT NULL,
address    VARCHAR(100) NOT NULL,
city       VARCHAR(50) NOT NULL);

CREATE TABLE Game(
game_id      DECIMAL(12) PRIMARY KEY,
season_id    DECIMAL(12) NOT NULL,
home_team_id DECIMAL(12) NOT NULL,
away_team_id DECIMAL(12) NOT NULL,
arena_id     DECIMAL(12) NOT NULL,
home_pog     DECIMAL(12) NOT NULL,
away_pog     DECIMAL(12) NOT NULL,
game_date DATE NOT NULL,
FOREIGN KEY (home_team_id) REFERENCES Team(team_id),
FOREIGN KEY (away_team_id) REFERENCES Team(team_id),
FOREIGN KEY (home_pog) REFERENCES Player(player_id),
FOREIGN KEY (away_pog) REFERENCES Player(player_id),
FOREIGN KEY (arena_id) REFERENCES Arena(arena_id));
```

```
CREATE TABLE Box_Score(
game_id         DECIMAL(12),
player_id       DECIMAL(12),
starter         VARCHAR(3) NOT NULL CHECK (starter IN('Yes', 'No')),
minutes         DECIMAL(3),
shot_made       DECIMAL(3),
shot_attempted  DECIMAL(3),
three_made      DECIMAL(3),
three_attempted DECIMAL(3),
ft_made         DECIMAL(3),
ft_attempted    DECIMAL(3),
o_rebound       DECIMAL(3),
d_rebound       DECIMAL(3),
assist          DECIMAL(3),
block           DECIMAL(3),
steal           DECIMAL(3),
personal_foul   DECIMAL(3),
turnover        DECIMAL(3),
point_score     DECIMAL(3),
plus_minus      DECIMAL(3),
CONSTRAINT bs_pk PRIMARY KEY(game_id, player_id),
FOREIGN KEY (game_id) REFERENCES Game(game_id),
FOREIGN KEY (player_id) REFERENCES Player(player_id));

CREATE TABLE Starter(
game_id   DECIMAL(12),
player_id DECIMAL(12),
CONSTRAINT starter_pk PRIMARY KEY(game_id, player_id),
FOREIGN KEY (game_id) REFERENCES Game(game_id),
FOREIGN KEY (player_id) REFERENCES Player(player_id));

CREATE TABLE Substitute(
game_id   DECIMAL(12),
player_id DECIMAL(12),
CONSTRAINT sub_pk PRIMARY KEY(game_id, player_id),
FOREIGN KEY (game_id) REFERENCES Game(game_id),
FOREIGN KEY (player_id) REFERENCES Player(player_id));

CREATE TABLE Trade(
player_id DECIMAL(12) NOT NULL,
old_team_id DECIMAL(12),
new_team_id DECIMAL(12),
trade_date DATE NOT NULL,
FOREIGN KEY (player_id) REFERENCES Player(player_id));

CREATE OR REPLACE TRIGGER trade_trg
BEFORE UPDATE ON Player
FOR EACH ROW
BEGIN
    IF :OLD.current_team_id <> :NEW.current_team_id THEN
    INSERT INTO Trade VALUES (:New.player_id, :OLD.current_team_id, :NEW.current_team_id, TRUNC(sysdate));
    END IF;
END;
/
```

The SQL script is also attached in the submission.

BasketballStats Indexing

As far as primary keys which are already indexed, here is the list.

Team.TeamID	NBA.NBAID	NCAA.NCAAID
G-League.G_LeagueID	Player.PlayerID	Game.GameID
BoxScore.GameID	BoxScore.PlayerID	Arena.ArenaID
Starter.GameID	Starter.PlayerID	Substitute.GameID
Substitute.PlayerID		

All of my foreign keys need an index, here is the list. They are all not unique.

Game.HomeTeamID	Game.AwayTeamID	Game.ArenaID
Game.HomePOG	Game.AwayPOG	Trade.PlayerID
Player.CurrentTeamID		

Three additional columns that are needed to be indexed:

Game.GameDate – it's reasonable that the date of game will be a limiting column in queries, because fans will want to check how the team or player has performed over the past week or month. This is a non-unique index because many games can be played on the same day.

Trade.TradeDate – fans will want to check the player's movement especially in the off-season. This is a non-unique index because many trades can happen on the same day.

BoxScore.3PTMade – The three-point shot is a key part of winning in the modern NBA, it's reasonable that there will be many queries that limit by three-point made, to see how many players or who had 3PTMade over a certain amount. This is a non-unique index because many players could make the same amount of 3-pointer in a game or in a season.

BasketballStats Index Creations

I named the index "gamehometeamIdx" to help identify what it's for, and placed the non-unique index on the home_team_id column in Game.

```
CREATE INDEX gamehometeamIdx
ON Game(home_team_id);
```

I named the index "gameawayteamIdx" to help identify what it's for, and placed the non-unique index on the away_team_id column in Game.

```
CREATE INDEX gameawayteamIdx
ON Game(away_team_id);
```

I named the index "gamearenaIdx" to help identify what it's for, and placed the non-unique index on the arena_id column in Game.

```
CREATE INDEX gamearenaIdx
ON Game(arena_id);
```

I named the index "GameHomePOGIdx" to help identify what it's for, and placed the non-unique index on the home_pog column in Game.

```
CREATE INDEX GameHomePOGIdx
ON Game(home_pog);
```

I named the index "GameAwayPOGIdx" to help identify what it's for, and placed the non-unique index on the away_pog column in Game.

```
CREATE INDEX GameAwayPOGIdx
ON Game(away pog);
```

I named the index "TradePlayerIDIdx" to help identify what it's for, and placed the non-unique index on the player_id column in Trade.

```
CREATE INDEX TradePlayerIDIdx
ON Trade(player_id);
```

I named the index "PlayerCurrentTeamIdx" to help identify what it's for, and placed the non-unique index on the current_team_id column in Player.

```
CREATE INDEX PlayerCurrentTeamIdx
ON Player(current_team_id);
```

I named the index "GameDateIdx" to help identify what it's for, and placed the non-unique index on the game_date column in Game.

```
CREATE INDEX GameDateIdx
ON Game(game_date);
```

I named the index "TradeDateIdx" to help identify what it's for, and placed the non-unique index on the trade_date column in Trade.

```
CREATE INDEX TradeDateIdx
ON Trade(trade date);
```

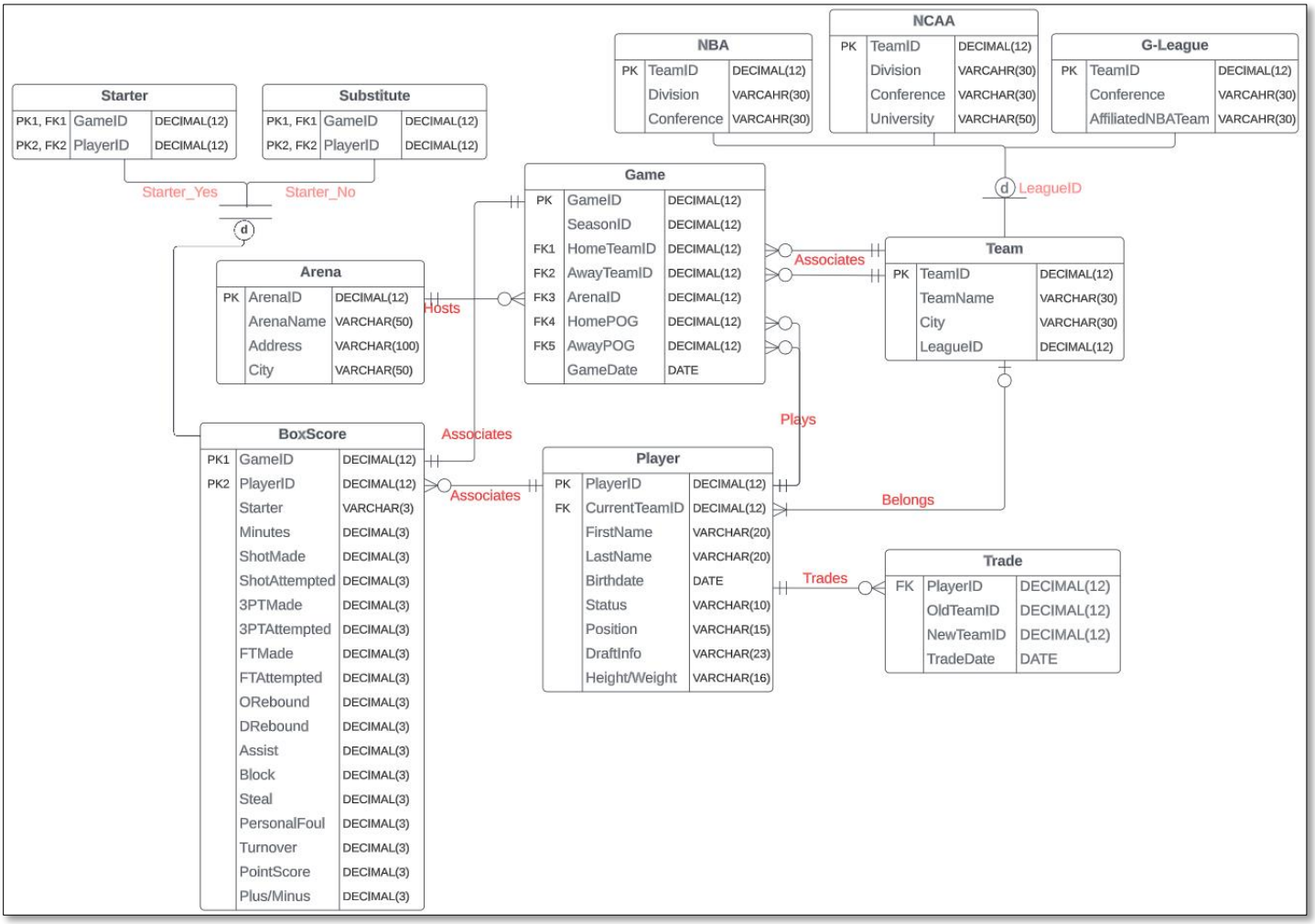
I named the index "BS3ptMadeIdx" to help identify what it's for, and placed the non-unique index on the three_made column in Box_Score.

```
CREATE INDEX BS3ptMadeIdx
ON Box_Score(three_made);
```

Basketball Database Term Project Iteration 5

Final ERD (From Iteration 4)

Here is the final look of my ERD:



BasketballStats Transaction

Team Record Use Case

1. A new team joins the NBA.
2. The BasketballStats staff enters the team information into the system, including team name, city, league, division and other important information.

```
--Team Record Use Case
CREATE OR REPLACE PROCEDURE addNBAteam (TEAM_ID IN DECIMAL, TEAM_NAME IN VARCHAR, CITY IN VARCHAR,
    DIVISION IN VARCHAR, CONFERENCE IN VARCHAR)
AS
BEGIN
    INSERT INTO Team(TEAM_ID, TEAM_NAME, CITY, LEAGUE_ID)
    VALUES(TEAM_ID, TEAM_NAME, CITY, 1);

    INSERT INTO NBA(TEAM_ID, DIVISION, CONFERENCE)
    VALUES(TEAM_ID, DIVISION, CONFERENCE);
END;
/

BEGIN
    addNBAteam(1, 'Celtics', 'Boston', 'Atlantic', 'Eastern');
    addNBAteam(2, '76ers', 'Philadelphia', 'Atlantic', 'Eastern');
    addNBAteam(3, 'Warriors', 'Golden State', 'Pacific', 'Western');
    addNBAteam(4, 'Lakers', 'Los Angeles', 'Pacific', 'Western');
    COMMIT;
END;
/
```

```
Script Output
Task completed in 0.053 seconds

PL/SQL procedure successfully completed.
```

I name the stored procedure “addNBAteam”, and give it parameters that correspond to the Team and NBA tables. Since this procedure is always for an NBA Team, I do not use a parameter for League_ID, but hardcode the number 1, representing NBA League. Inside the stored procedure, there are two insert statements to insert into the two respective tables.

I added four teams into the database.

Player Profile Use Case

1. A new basketball player joins one of the recorded teams.
2. The BasketballStats team enters the new player's information into the system, including name, birthdate, position, height/weight, and other important information.

```
--Player Profile Use Case
CREATE OR REPLACE PROCEDURE addplayer(PLAYER_ID IN DECIMAL, CURRENT_TEAM_ID IN DECIMAL,
    FIRST_NAME IN VARCHAR, LAST_NAME IN VARCHAR, BIRTHDATE IN DATE, STATUS IN VARCHAR,
    POSITION IN VARCHAR, DRAFT_INFO IN VARCHAR, HEIGHT_WEIGHT IN VARCHAR)
AS
BEGIN
    INSERT INTO Player(PLAYER_ID, CURRENT_TEAM_ID, FIRST_NAME, LAST_NAME, BIRTHDATE, STATUS, POSITION, DRAFT_INFO, HEIGHT_WEIGHT)
    VALUES (PLAYER_ID, CURRENT_TEAM_ID, FIRST_NAME, LAST_NAME, BIRTHDATE, STATUS, POSITION, DRAFT_INFO, HEIGHT_WEIGHT);
END;
/
BEGIN
    addplayer(1, 1, 'Jayson', 'Tatum', CAST('03-MAR-1998' AS DATE), 'Healthy', 'Small Forward',
    '2017: Rd 1, Pk 3 (BOS)', '6'' 8'''', 210 lbs');
    addplayer(2, 2, 'James', 'Harden', CAST('26-AUG-1989' AS DATE), 'Healthy', 'Shooting Guard',
    '2009: Rd 1, Pk 3 (OKC)', '6'' 5'''', 220 lbs');
    addplayer(3, 1, 'Jaylen', 'Brown', CAST('24-OCT-1996' AS DATE), 'Healthy', 'Shooting Guard',
    '2016: Rd 1, Pk 3 (BOS)', '6'' 6'''', 223 lbs');
    addplayer(4, 1, 'Al', 'Horford', CAST('03-JUN-1986' AS DATE), 'Healthy', 'Center',
    '2007: Rd 1, Pk 3 (ATL)', '6'' 9'''', 240 lbs');
    addplayer(5, 1, 'Derrick', 'White', CAST('02-JUL-1994' AS DATE), 'Healthy', 'Point Guard',
    '2017: Rd 1, Pk 29 (SA)', '6'' 4'''', 190 lbs');
    addplayer(6, 1, 'Marcus', 'Smart', CAST('06-MAR-1994' AS DATE), 'Healthy', 'Point Guard',
    '2014: Rd 1, Pk 6 (BOS)', '6'' 4'''', 220 lbs');
    addplayer(7, 2, 'Joel', 'Embiid', CAST('16-MAR-1994' AS DATE), 'Healthy', 'Center',
    '2014: Rd 1, Pk 3 (PHI)', '7'' 0'''', 280 lbs');
    addplayer(8, 2, 'P.J.', 'Tucker', CAST('05-MAY-1985' AS DATE), 'Healthy', 'Power Forward',
    '2006: Rd 2, Pk 35 (TOR)', '6'' 5'''', 245 lbs');
    addplayer(9, 2, 'Tobias', 'Harris', CAST('15-JUL-1992' AS DATE), 'Healthy', 'Power Forward',
    '2011: Rd 1, Pk 19 (CHA)', '6'' 7'''', 226 lbs');
    addplayer(10, 2, 'Tyrese', 'Maxey', CAST('04-NOV-2000' AS DATE), 'Healthy', 'Point Guard',
    '2020: Rd 1, Pk 21 (PHI)', '6'' 2'''', 200 lbs');
    COMMIT;
END;
/

Script Output x  Query Result x
Task completed in 0.111 seconds

PL/SQL procedure successfully completed.
```

Game Result Use Case

1. A basketball game is over (for example, an NBA game).
2. The BasketballStats team enters the game stats into the system, including home and away team's information, arena id, players of the game, and other important information.

```
--GAME RESULT USE CASE
--Before running the following stored procedure:
INSERT INTO arena VALUES (1, 'TD Garden', '100 Legends Way', 'Boston');
INSERT INTO ARENA VALUES (2, 'Wells Fargo Center', '3601 S Broad St', 'Philadelphia');

CREATE OR REPLACE PROCEDURE AddGame (game_id IN DECIMAL, season_id IN DECIMAL, home_team_id IN DECIMAL, HOME_TEAM_PT IN DECIMAL,
away_team_id IN DECIMAL, AWAY_TEAM_PT IN DECIMAL, arena_id IN DECIMAL, home_pog IN DECIMAL, away_pog IN DECIMAL)
AS
BEGIN
    INSERT INTO Game
    VALUES(GAME_ID , SEASON_ID, HOME_TEAM_ID, HOME_TEAM_PT, AWAY_TEAM_ID, AWAY_TEAM_PT, ARENA_ID, HOME_POG, AWAY_POG, TRUNC(sysdate));
END;
/

BEGIN
    addgame(1, 77, 1, 126, 2, 117, 1, 1, 2);
    addgame(2, 77, 1, 106, 2, 99, 1, 5, 7);
    addgame(3, 77, 2, 107, 1, 110, 2, 7, 3);
    addgame(4, 77, 2, 103, 1, 101, 2, 7, 5);
    addgame(5, 77, 1, 115, 2, 119, 1, 1, 2);
    addgame(6, 77, 1, 121, 2, 87, 1, 3, 9);
    addgame(7, 77, 2, 102, 1, 114, 2, 7, 1);
    addgame(8, 77, 2, 116, 1, 115, 2, 2, 1);
    addgame(9, 77, 1, 103, 2, 115, 1, 1, 7);
    addgame(10, 77, 2, 86, 1, 95, 2, 7, 6);
    addgame(11, 77, 1, 112, 2, 88, 1, 1, 9);
    COMMIT;
END;
/
```

Script Output | Query Result

Task completed in 0.111 seconds

```
PL/SQL procedure successfully completed.
```

BasketballStats History

I actually already created a trade history table in ERD and wrote a trigger in iteration 4. But I modified the trigger slightly based on the explanation document. The Trade History entity is linked to Player. Below are its attributes (From iteration 4).

Trade	Player ID, Old Team ID, New Team ID	DECIMAL(12)	After completing lab 4, I decided to add a history table to collect player's team changes.
Trade	Trade Date	DATE	The date a player is getting traded.

Here is a screenshot of my table creation.

```
--BasketballStats History (Trade History Trigger)
CREATE TABLE Trade(
trade_id DECIMAL(12) PRIMARY KEY,
player_id DECIMAL(12) NOT NULL,
old_team_id DECIMAL(12),
new_team_id DECIMAL(12),
trade_date DATE NOT NULL,
FOREIGN KEY (player_id) REFERENCES Player(player_id));
```

Script Output | Task completed in 0.072 seconds

```
Table TRADE created.
```

Here is a screenshot of my trigger creation.

```
--(Trade History Trigger)
CREATE OR REPLACE TRIGGER trade_trg
BEFORE UPDATE OF CURRENT_TEAM_ID ON Player
FOR EACH ROW
BEGIN
    INSERT INTO Trade VALUES (NVL((SELECT MAX(trade_id)+1 FROM Trade), 1),
                              :New.player_id,
                              :OLD.current_team_id,
                              :NEW.current_team_id,
                              TRUNC(sysdate));
END;
/
```

Script Output | Task completed in 0.405 seconds

```
Trigger TRADE_TRG compiled
```

I start by ensuring there is a Player table created. It has several rows of data since we created and ran a stored procedure in the previous section.

```
SELECT * FROM PLAYER;
```

Script Output | Query Result | SQL | All Rows Fetched: 2 in 0.009 seconds

	PLAYER_ID	CURRENT_TEAM_ID	FIRST_NAME	LAST_NAME	BIRTHDATE	STATUS	POSITION	DRAFT_INFO	HEIGHT_WEIGHT
1	1	1	Jayson	Tatum	03-MAR-98	Healthy	Small Forward	2017: Rd 1, Pk 3 (BOS)	6' 8'', 210 lbs
2	2	2	James	Harden	26-AUG-89	Healthy	Shooting Guard	2009: Rd 1, Pk 3 (OKC)	6' 5'', 220 lbs

Next, I hypothetically trade James Harden to the Los Angeles Lakers and then trade him back to the Philadelphia 76ers.

```
199  UPDATE Player SET current_team_id = 4 WHERE player_id = 2;
200  UPDATE Player SET current_team_id = 2 WHERE player_id = 2;
```

Script Output | Query Result | Task completed in 0.056 seconds

```
1 row updated.
```

Last, I verify that Trade table has a record of these changes.

```
202  SELECT * FROM TRADE;
```

Script Output | Query Result | SQL | All Rows Fetched: 2 in 0.003 seconds

	TRADE_ID	PLAYER_ID	OLD_TEAM_ID	NEW_TEAM_ID	TRADE_DATE
1	1	2	2	4	06-AUG-23
2	2	2	4	2	06-AUG-23

Notice that there are two rows, one for the change from team 2 (PHI) to team 4 (LAL), and the second from team 4 back to team 2. Trades are now tracked with a trigger and a history table.

BasketballStats Question and Query

- What was Boston's average points scored when they won against Philadelphia? (Since I only inserted BOS-PHI games into the database).

```
238  --BOS avg points
239  SELECT w_team, AVG(w_score)
240  FROM(
241      SELECT game_id, home.city||' '||home.team_name AS W_team, home_team_pt AS w_score
242      FROM game
243      JOIN team home ON game.home_team_id = home.team_id
244      WHERE home.city = 'Boston' AND home_team_pt > away_team_pt
245      UNION ALL
246      SELECT game_id, away.city||' '||away.team_name AS W_team, away_team_pt AS w_score
247      FROM game
248      JOIN team away ON game.away_team_id = away.team_id
249      WHERE away.city = 'Boston' AND home_team_pt < away_team_pt
250      ORDER BY game_id)
251  GROUP BY w_team;
```

Script Output | Query Result | SQL | All Rows Fetched: 1 in 0.029 seconds

	W_TEAM	AVG(W_SCORE)
1	Boston Celtics	112

Used techniques: JOINs, WHEREs, ORDER, aggregate function, subquery, union.

I start by retrieving all the games Boston won at home and away, then calculate the average points scored. To prove the query result is correct, here is a screenshot of the subquery's result and the Game table.

	GAME_ID	W_TEAM	W_SCORE
1	1	Boston Celtics	126
2	2	Boston Celtics	106
3	3	Boston Celtics	110
4	6	Boston Celtics	121
5	7	Boston Celtics	114
6	10	Boston Celtics	95
7	11	Boston Celtics	112

	GA...	SEASON_ID	HOME_TEAM_ID	HOME_TEAM_PT	AWAY_TEAM_ID	AWAY_TEAM_PT
1	1	77	1	126	2	117
2	2	77	1	106	2	99
3	3	77	2	107	1	110
4	4	77	2	103	1	101
5	5	77	1	115	2	119
6	6	77	1	121	2	87
7	7	77	2	102	1	114
8	8	77	2	116	1	115
9	9	77	1	103	2	115
10	10	77	2	86	1	95
11	11	77	1	112	2	88

- The Player of the Game award is awarded to the player with the highest score for each team. Who won the most PoG last season in BOS-PHI games?

```
262 SELECT player, COUNT(player) AS pog_times
263 FROM (
264     SELECT home_pog AS id, h.first_name||' '||h.last_name as player
265     FROM game
266     JOIN player h ON game.home_pog = h.player_id
267     UNION ALL
268     SELECT away_pog AS id, a.first_name||' '||a.last_name as player
269     FROM game
270     JOIN player a ON game.away_pog = a.player_id)
271 GROUP BY player
272 ORDER BY -pog_times;
```

Script Output | Query Result | SQL | All Rows Fetched: 7 in 0.202 seconds

	PLAYER	POG_TIMES
1	Jayson Tatum	6
2	Joel Embiid	6
3	James Harden	3
4	Derrick White	2
5	Jaylen Brown	2
6	Tobias Harris	2
7	Marcus Smart	1

Used techniques: JOINs, ORDER, COUNT, subquery, union.

I wrote two queries to get the name of all home_POG and away_POG respectively, then combined them as a single table using UNION clause. Finally, I wrote an outer query to count the number of POGs each player received last season.

- What player moves (TRADE) have been made with the Boston Celtics over the past 3 months?

```
288 SELECT first_name||' ' ||last_name AS player,
289     old_team.city||' '||old_team.team_name AS old_team,
290     new_team.city||' '||new_team.team_name AS new_team,
291     trade_date
292 FROM trade
293 JOIN player ON trade.player_id = player.player_id
294 JOIN team old_team ON old_team.team_id = trade.old_team_id
295 JOIN team new_team ON new_team.team_id = trade.new_team_id
296 WHERE trade_date > add_months(SYSDATE,-3) AND (old_team_id =1 OR new_team_id =1);
```

Script Output | Query Result | Query Result 1 | Query Result 2

SQL | All Rows Fetched: 3 in 0.002 seconds

	PLAYER	OLD_TEAM	NEW_TEAM	TRADE_DATE
1	Marcus Smart	Boston Celtics	Memphis Grizzlies	07-AUG-23
2	Kristaps Porzingis	Washington Wizards	Boston Celtics	07-AUG-23
3	Grant Williams	Boston Celtics	Dallas Mavericks	07-AUG-23

Used techniques: JOINs, WHEREs

The question regarding the history (TRADE) table is relatively easy. In order to provide more data for this, I added a couple more players and trades. The WHERE clause is relatively important in this query. I combined AND and OR conditions in WHERE. OR is used to filter transactions involving Boston, either old_team or new_team is Boston (BOS id is 1 in this case). '> add_months(SYSDATE,-3)' is used to filter trades that is happened last three months. Combining these two conditions requires AND.