Universidade do Porto

FEUP Faculdade de Engenharia

NBA

Curricular Unity: Databases
2017/2018

Group 1 from Class 3:

up201604184 - Helena Montenegro

up201604470 - Pedro Miguel Silva

up201703444 - Scutelnicu Petru

Contents

1. Context

The National Basketball Association needs a platform to save all the data related to one season.

There are many teams, formed by its players and its coach. Each team represents a state and a city, in which there is an arena defined by a name and its capacity.

It's important to know the players and the coaches' name, age and country.

The players also have their height, weight, position and t-shirt number. The players get selected to compete in teams for the NBA through the draft. It's also important to know the year they made it to the draft as well as their rank in it.

Both the players and the coaches can change teams in the middle of the season. There's a need to keep track of the date when they start playing for the team, as well as the coach's role in the team, which can be of assistant coach or head coach.

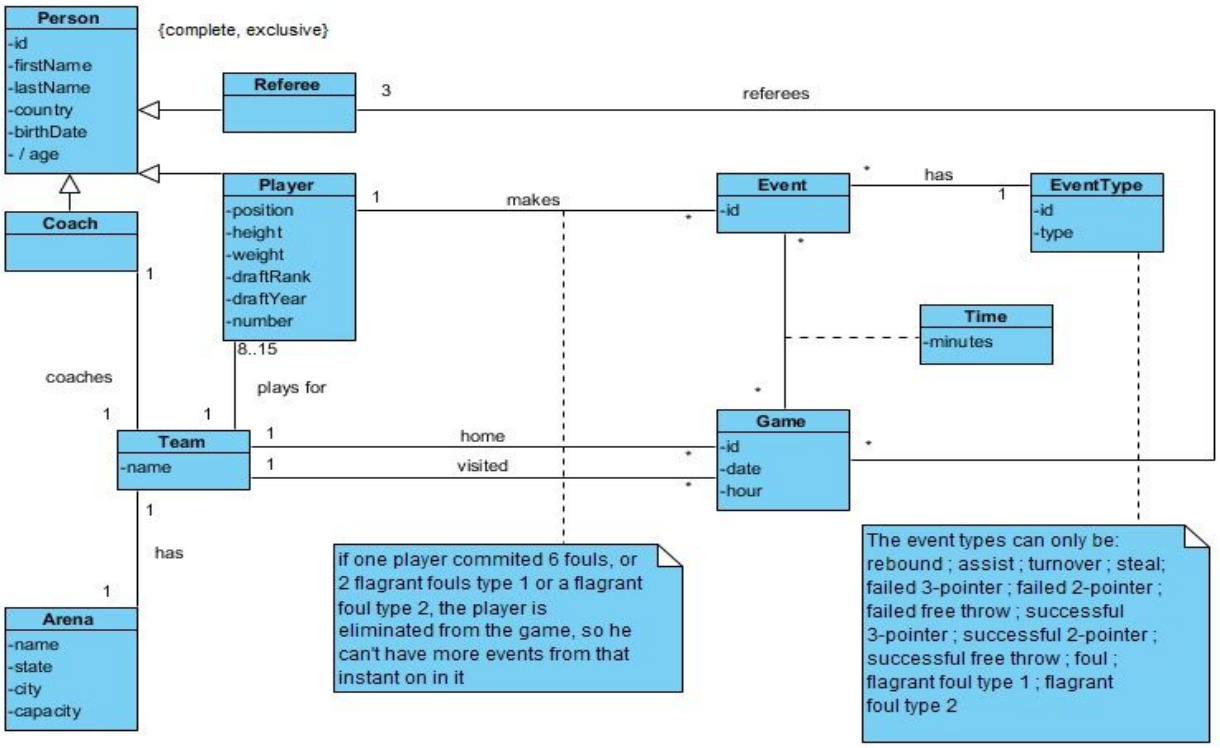
The season is composed by games, which are played between two teams. The games are characterized by a date and an hour and are played in one of the team's arena.

Each game has three referees, with an id, name, age and country.

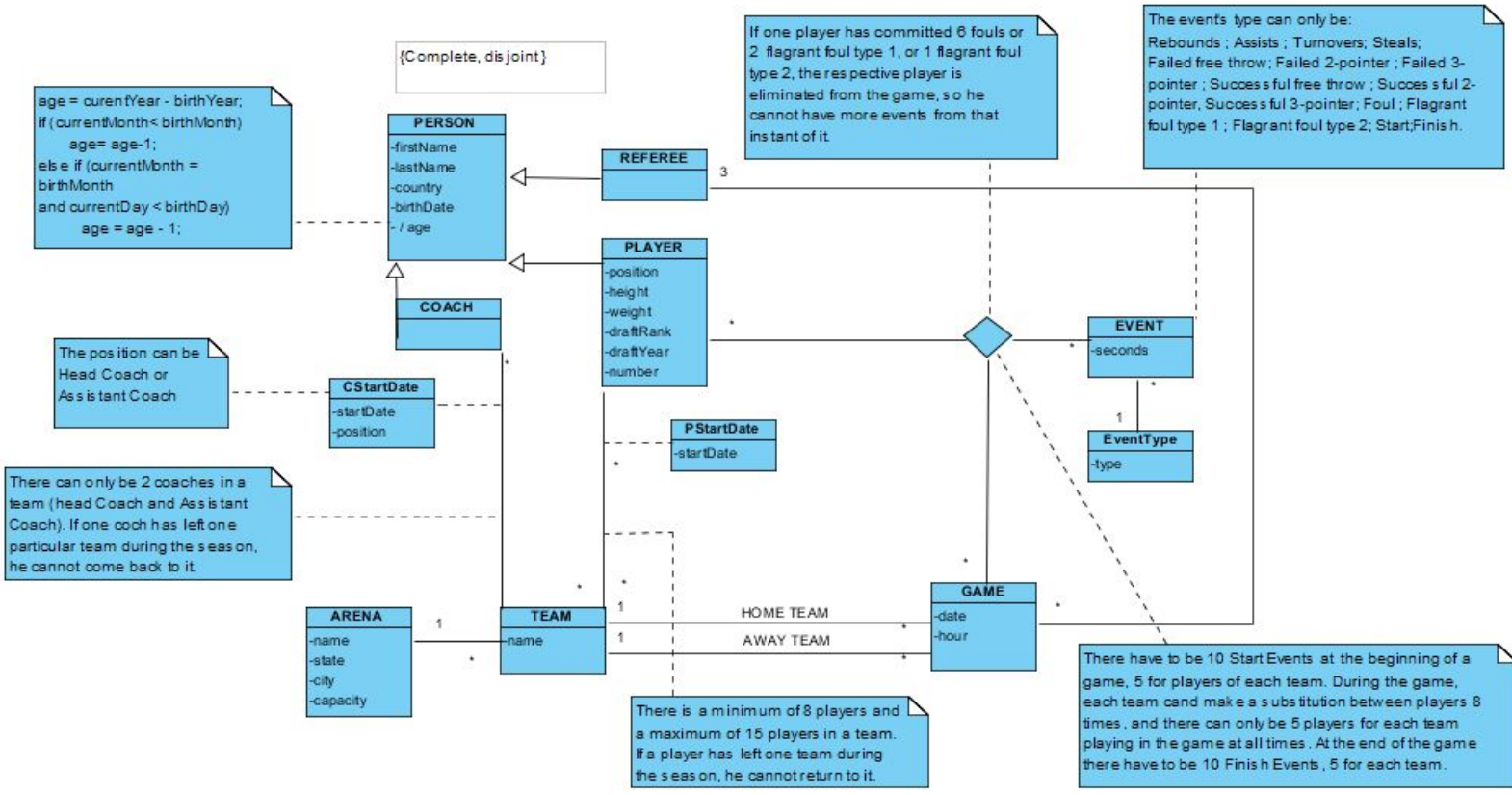
The main goal of our database is to keep track of every shot, foul, rebound, assist and other events made by each player in each instance of the game. This information will be used to create statistics for each player and each team, which can be used in multiple ways, like for a team to study their opponents before their games or for public's access.

2. UML Diagram

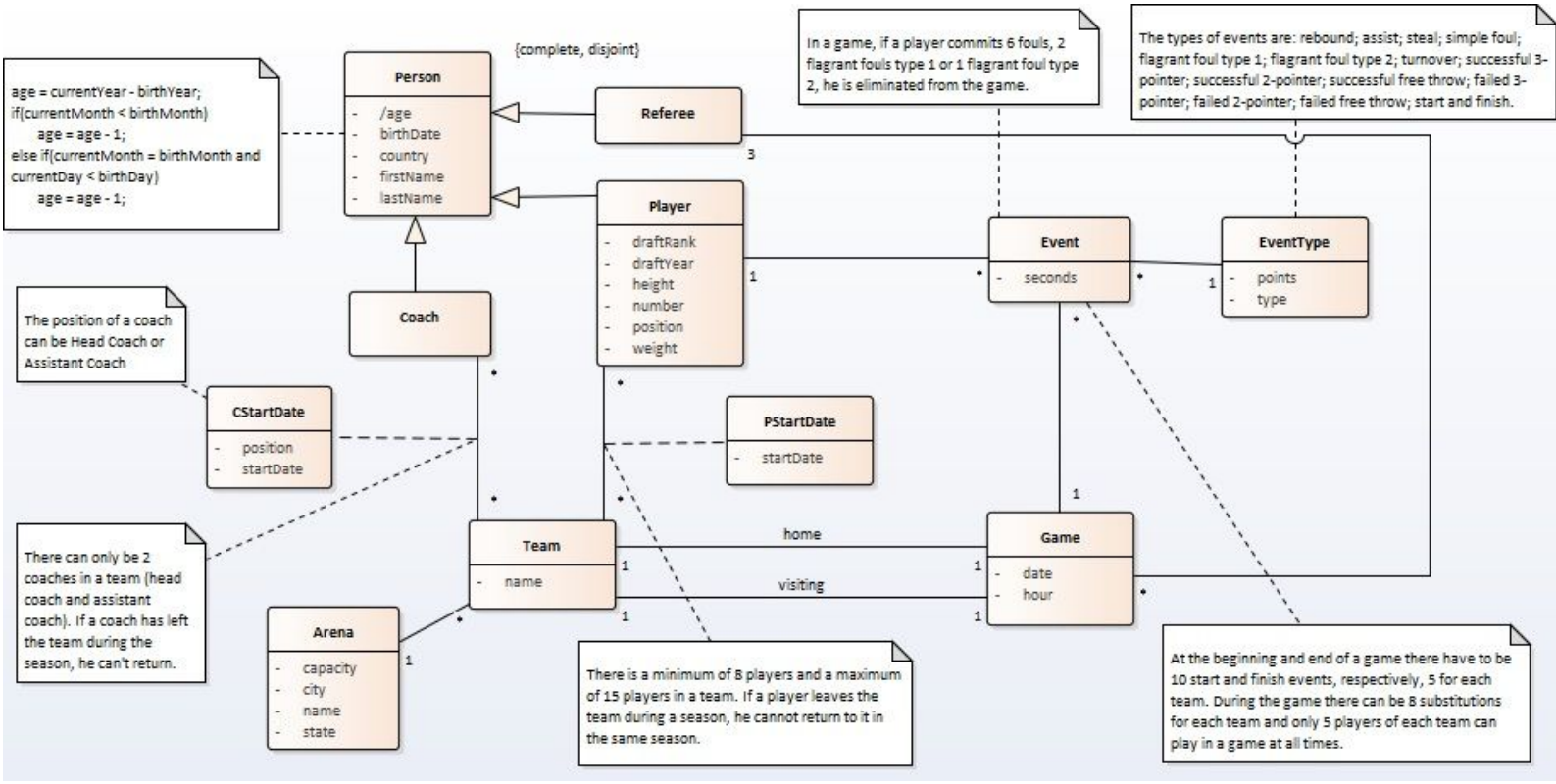
First delivery:



Second delivery:



Final delivery:



2.1. UML Diagram's Explanation

There were common attributes between the classes Coach, Referee and Player, therefore we put all of these in a superclass (Person). The resulting generalization is complete and disjoint.

To keep track of every rebound, foul, assist or shot done in a game, we created a class Event, associated with a game and a player, in a ternary association, with the class EventType, and that has as an attribute the number of seconds into the game when the event was made. Thanks to this, we know in each instant of the game if a determined type of event was made and who made it. Knowing the number of points each player scored in the game, we can know the full score for each team and deduce which team won the game. The events will also contribute to the players' personal stats.

The types of events are: rebound; assist; steal; simple foul; flagrant foul type 1; flagrant foul type 2; turnover; successful 3-pointer; successful 2-pointer; successful free throw; failed 3-pointer; failed 2-pointer; failed free throw; start and finish (entering and leaving the game respectively).

A player with 6 fouls, 2 flagrant fouls type 1 or 1 flagrant foul type 2 has to be substituted by a team mate and can’t play for the rest of the game.

We divided the association between team and game in two associations in order to differentiate in which team’s arena the game will be played.

3. Relational Model

Coach(idCoach, firstName, lastName, country, birthDate, age)

Referee(idReferee, firstName, lastName, country, birthDate, age)

Player(idPlayer, firstName, lastName, country, birthDate, age, position, height, weight, draftRank, drafYear, number)

Team(idTeam, name, arena->Arena)

Arena(name, state, city, capacity)

CStartDate(idCoach->Coach, team->idTeam, date, position)

PStartDate(idPlayer->Player, team->idTeam, date)

Game(idGame, date, hour, homeTeam->idTeam, visitingTeam->idTeam)

RefereeGame(referee->Referee, game->Game)

Event(game->Game, player->Player, type->EventType, seconds)

EventType(idType, type, points)

We decided to add ids for the relations which will be repeated more often, in order to save memory space, since an integer occupies less space than a string or than a compound key.

The generalization is complete and disjoint which means that every object of the class Person belongs to one and only one of the subclasses. For this reason we used the object oriented style, where the class Person is not represented.

3.1. Functional Dependencies and Normal Forms Analysis

The relations not represented under the following subtitles have trivial functional dependencies.

3.1.1. Coach

Coach(idCoach, firstName, lastName, country, birthDate, age)

1) idCoach-> firstName, lastName, birthDate, country
2) birthDate->age
3) firstName, lastName, birthDate, country -> idCoach

The second functional dependency (birthDate->age) violates the Boyce-Codd Normal Form, since birthDate is not a superkey of the relation. To put this relation in this form, we could divide it in two relations:

BirthDate(birthDate, age)

Coach(idCoach, firstName, lastName, country, birthDate->BirthDate)

The relation BirthDate only has the second functional dependency and the relation Coach has the remaining ones, which results in having no more violations to either the BCNF or the Third Normal Form. In the implementation of the database, we decided not to implement the relation BirthDate, since it wasn't important enough and because age is a derived element.

3.1.2. Referee

Referee(idReferee, firstName, lastName, country, birthDate, age)

1) idReferee -> firstName, lastName, birthDate, country, age
2) birthDate->age
3) firstName, lastName, birthDate, country, age -> idReferee

This situation is exactly the same has the one in Coach. By having the BirthDate relation, there would be no more violations.

3.1.3. Player

Player(idPlayer, firstName, lastName, country, birthDate, age, position, height, weight, draftRank, drafYear, number)

1) idPlayer -> firstName, lastName, birthDate, country, position, height, weight, draftRank, drafYear, number, age
2) birthDate->age
3) firstName, lastName, birthDate, country, position, height, weight, draftRank, drafYear, number, age -> idPlayer

This situation is exactly the same has the one in Coach. By having the BirthDate relation, there would be no more violations.

3.1.4.Team

Team(idTeam, name, arena->Arena)

1) idTeam->name, arena
2) name, arena -> idTeam

In this relation, there are no violations to the Boyce-Codd Normal Form since the left side of the functional dependencies are the keys to the relation. For the same reason, there are also no violations to the Third Normal Form.

3.1.5. Arena

Arena(name, state, city, capacity)

1) name -> state, city, capacity

Since there's only one functional dependency where the left side is the primary key of the relation, there are no violations to either the third normal form or the Boyce Codd normal form.

3.1.6. CStartDate

CStartDate(idCoach->Coach, team->Team, date, position)

1) idCoach, team -> date, position

The relation is in the Boyce-Codd Normal Form, since the only functional dependency has in its left side the composed key of the relation. It is also in the third normal form.

3.1.7. PStartDate

PStartDate(idPlayer->Player, team->Team, date)

1) idPlayer, team -> date

The relation is in the Boyce-Codd Normal Form, since the only functional dependency has in its left side the composed key of the relation. It is also in the third normal form.

3.1.8. Game

Game(idGame, date, hour, homeTeam->Team, visitingTeam->Team)

1) idGame -> hour, date, homeTeam, visitingTeam
2) hour, date, homeTeam, visitingTeam -> idGame

There are no violations to the Boyce-Codd Normal Form. Since the functional dependencies have a key on its left side and the relation represents all the attributes present in it, there are also no violations to the third normal form.

3.1.9. EventType

EventType(idType, type, points)

1) idType -> type, points
2) type -> idType, points

There are no violations to either the Boyce Codd normal form or the third normal form since the functional dependencies of the relation have, on the left side, the primary key of the relation.

4. Restrictions

- Every attribute but the draftYear and draftRank in Player has to have some value. This can be implemented by putting the restriction NOT NULL in every attribute but the ones mentioned. If a player has draftYear and draftRank as null, it means that they haven't entered the draft yet, having no rank nor year in the draft as a result.
- The seconds, in Event have to be a positive number. This can be solved by having the condition CHECK(seconds>=0) in this attribute.
- The elements underlined in the relational model have to be primary keys, which can be done with the restriction PRIMARY KEY.
- The foreign keys also have to be implemented by using the word REFERENCES.
- The position of a coach in a team can either be "Assistant Coach" or "Head Coach", which can be implemented with the following: CHECK (position = 'Assistant Coach' or position = 'Head Coach').
- In the class Game, the teams playing cannot be the same, which can be solved with the restriction: CHECK(homeTeam <> visitingTeam).
- One team can't have more than a game in the same date and hour, which can be solved by declaring as UNIQUE the combination of date, hour and homeTeam, and the combination of date, hour and visitingTeam.
- The height and the weight of a player have to be more than zero, which can be solved with the conditions: CHECK(height>0); CHECK(weight>0).
- There can't be two players with the same draftYear and draftRank, therefore, we can declare the combination of there as UNIQUE.
- There can't be two teams with the same name, therefore this attribute has to be declared as UNIQUE.
- In the relation Event, there can't be two events of the same type in the same second, therefore we need to declare as UNIQUE the combination of these two.

5. Restrictions that need a trigger

- If a player has 6 fouls, 2 flagrant fouls type 1 or a flagrant foul type 2 in one game then it is substituted by another player of the team and is not allowed back in that game.
- An assist implies that a shot was made by someone in the team briefly after (the player who assists can't be the player who scores).
- A rebound implies that there was a missed shot briefly before.
- A free throw implies that one player in the other team made a foul before (flagrant or not).
- A steal implies that one player in the opposing team had a turnover.
- At the beginning of the game (seconds = 0) there has to be events of type Start for 5 distinct players of each team and at the end of the game, there have t be 5 events of type Finish.
- A Finish in the middle of a game implies a Start for another player, and vice-versa.

6. List of Queries Implemented

1. Date and hour of the games that have already been played, with the teams that played in the game as well as their respective score.
2. List of games not yet played, represented by their date and hour, as well as the teams playing.
3. List of the names and teams of the players that have played in every game of the season where their team entered.
4. Events and the average height of the players that made the event, as well as the height of the tallest player and the shortest one.
5. Top 3 players that scored the most points during the season. In order to make the query a little harder we didn't use any instructions to select the three first players from the table that has the players and the points each player did. Instead we used the

instruction max() in order to get the three players which should be first and then we only used the order by to be certain the players were in the right order.

6. Countries that had players that made the most of each type of event, and how many players of the country did that event.
7. List of players that were eliminated from the game, as well as their teams, the reason for elimination and the game they were eliminated from.
8. List of groups of referees with respective number of fouls given during the season.
9. List of teams and the number of the games they played, ordered by the number of wins. In this query we reused the first query in this list. We have two subqueries in the FROM clause, one to check how many games each team as been in and the other to check how many games the team won, where we reused the first query.
10. Name of the arenas where there have been games played, as well as the event that was done more times there, ordered by the diversity of the events that happened in the arena.

7. List of Triggers Implemented

1. InvalidEvents: Trigger that does not allow any insert of any event made by a player who is not at the game at the time the event was made. After every insert, the trigger will check if the player making the event is in the game, by checking if there's a start event before the inserted event and if there's no finish or expelled event in between the closest start event and the new event. The trigger will also check if the player belongs to one of the teams playing in the game.
2. Elimination: Trigger that expels a player from the game by inserting an event of type "Expelled" if the player has achieved his sixth foul or his second flagrant foul type 1 or a flagrant foul type 2. The new tuple will have the same idPlayer and idGame, as the inserted one and the seconds will be the same as the latest foul of the player committed in the game.
3. NumberOfPlayers: Trigger that makes sure that in each instance of the game there's a maximum of 5 players of each team at all instances.