Part 3: Project Report

Group 16

Fidelio Ciandy

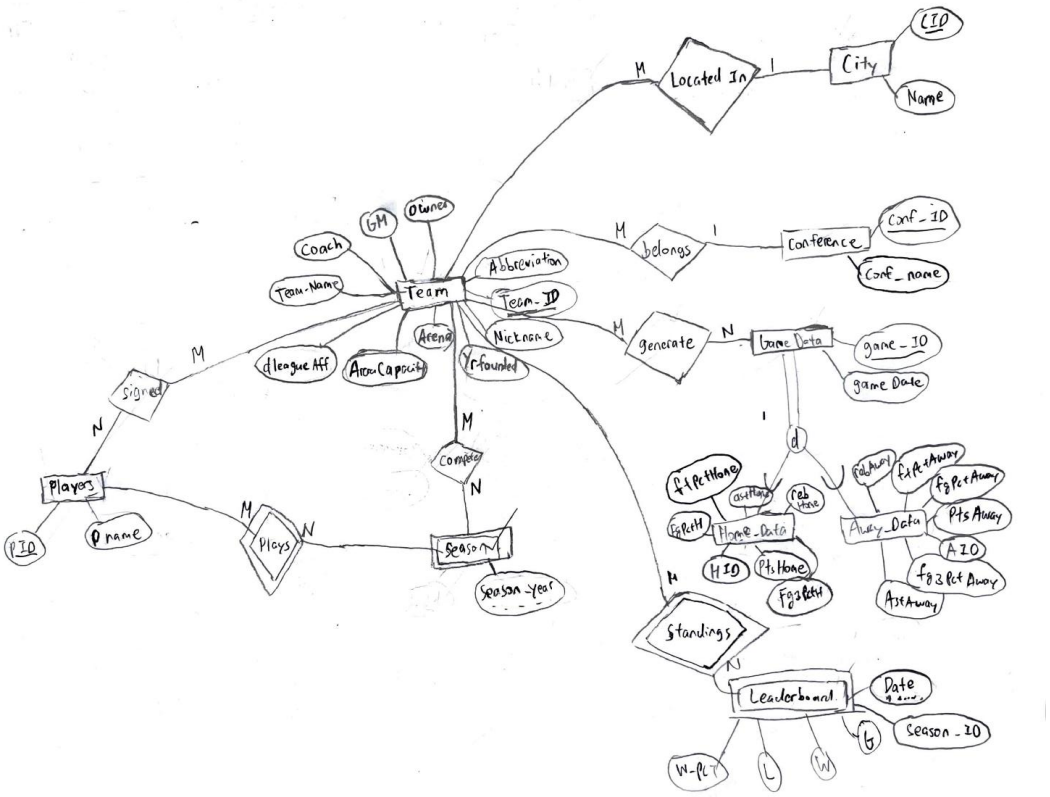
Yirong Wang

Summary of the data:

It was chosen because we are both NBA fans. Not only because we are NBA fans but also because this dataset has good quality and a huge amount of data. Also, this dataset has attributes that are familiar to us. Therefore, it is easier for us to interpret and understand it.

At first, the original data (https://www.kaggle.com/datasets/nathanlauga/nba-games) consists of five tables, we decided to remove one table (game_details.csv) because it contains a lot of data that has too many game-specific details and a huge amount of data, which would have resulted in a complex final model. Furthermore, we split those five tables into ten tables that have been normalized into 3NF or BCNF.

As mentioned before, the original data set has 5 tables and we chose 4 of them to work with, here is the list of the files: player.csv with 4 columns and 7229 rows (265.58 kB), games.csv with 19 columns and 25k rows (3.99 MB), team.csv with 13 columns and 31 rows (4.06 kB), ranking.csv with 11 columns and 200k rows (14.83 MB). On the other hand, the ten tables that we have broken into consist of, city.csv with 29 rows (1 kB), compete.csv with 7228 rows (169 kB), conference.csv with 2 rows (1 kB), gameData.csv with 25k (2451 kB), generate.csv with 51k rows(1057 kB), leaderboard.csv with 120k rows (4953 kB), player.csv with 1770 rows (39 kB), season.csv with 6305 rows (80 kB), signed.csv 4281 rows (79 kB), team.csv with 30 rows (4 kB).



Discussion of the data model:

In an attempt to normalize the dataset, it was divided into tables, which would simplify management and lookup of the dataset. Normalizing the table will also decrease the redundancy of data. We began by inspecting all the attributes from the tables and making an attempt to remove any unnecessary columns and rows, such as game status being Final for all the games because all the games have ended. After that, we began by locating every single attribute from the original dataset. From the available attributes, we decided to make functional dependencies for each table and normalize them further into BCNF. Furthermore, we separate the table based on the normalized dependencies. We separated each table because each of them has a specific meaning and requires its own table. Through the process, we separate the season from the players.csv. Here we found that the season between each file is not the same. Therefore, we decided to limit the season from 2009 until 2019. It allows us to look up a specific season for a team and player. We also separated the city and conference from the Team table so that there won't be any repeated values on the

table. We also created new tables that represent many-to-many relationships in order for us to associate rows or columns of one table with another.

Designing the data model was really tricky and we did have difficulties along the way. For instance, on the leaderboard table, it was hard for us to determine whether it was a weak entity or not. At first, we thought that the standingsDate was the primary key for the table, but we were wrong. Turns out that leaderboard is a weak entity and the primary keys are standingsDate and team_id, it needs the primary key from the team table (team_id), to support the standingsDate attribute.

The data model that we have created cleanly fits into a relational database because the data is structured and the tables are related to one another. Our data model also has a primary key for each table which is an important thing relational databases have. The player, team, and other data are connected with each other through relationships and each record can be identified with a unique ID/key.

Furthermore, we did regret a few decisions that we have made regarding the model. For instance, not include the game detail.csv file in our dataset. If we include that table, we will have more detail on how individual players perform instead of only the team performance alone. We did change our data model a little from part 1. As we got deeper into the class content, we realized that a part of our model in part 1 is not really making any sense. For example, we created the relationship between season and team so we can access the team performance by each season. Beforehand, there is no way of doing it. We also fix our data model based on part 1 feedback, for example, the relationship between the conference and team should be one-to-many instead of many-to-many. Furthermore, we also decided not to write extra attributes in the relationship on the ER diagram.

Could the data be modeled in a different way:

We don’t think that the data can be modeled completely differently, maybe it could have some minor changes. For instance, we could join the city and team table since we can determine the city with the team_ID. We could also separate the home_data and away_data into a different file or table. But the main focus will still be on Team and their information. Given the work completed, we will still choose this model over other models.

Interesting queries and interface:

The queries that we have created will be displayed in an interface called Graphical User Interface (GUI). We are using Java Swing Library to create the GUI. The interface is pretty simple and will be fairly easy for users to use it. The user will click buttons to run the queries, and a prompt asking for the user input will be shown according to the query that is selected. The output of the query will be displayed in a form of a table.

One of the most interesting queries that we have, is the one that detects each team's regular or pre-season final records. The interface will allow the user to insert a team’s name and it will return the team’s name, season id, total games, games won, games lost, and win percentage in a form of a table. It is interesting because regular and pre-season records appear on the same table (leaderboard.csv) and it is hard for people to differentiate the statistics of the records if they are not sorted or selected specifically. Also, it is interesting because when people run the query, people will find out that in 2011, the NBA only allowed teams to play until 66 games in the regular season, which is weird because usually, teams will play a total of 82 games each season. This has a story behind it and that is what makes the query interesting.

Another interesting query is about players that never change any teams, it will return the names of the players. Knowing Stephen Curry never changed his team in his career, we want to see if other players were able to achieve this. With the limitation of the dataset, it will only show players that didn’t change teams between 2004 to 2020, but it was an interesting result that many players didn’t change their teams in this time period.

Does this dataset require a relational database?

Yes, because it's a well-structured table dataset, it will be the best option to use a relational database. Even though it is possible to use another database model, for instance, the graph database, it won't be any better because there will be way too many nodes in the graph and it will be hard to represent each game's data. On the other hand, if we use a relational database, we can easily see that game's data only in one table. We can easily analyze the table. Furthermore, the interesting queries will be harder if we were using an alternative database because it doesn't have any properties that relational databases have. For instance, using a relational database allows us to join and group the tables, we can also use set theory between tables easily. Therefore, to run queries easily and display data effectively it is better to use a relational database for this dataset.

Would this database be a good teaching tool for COMP 3380?

Yes and no. Yes because it has a fair amount of relationships and entities. Also, it's a fairly common concept for people, they can understand it fastly what is in each table and it will be easy to teach SQL syntax and query around it. No, because it has a large amount of data (one table with 120k+ rows). It will be hard for students if they want to see if their result is right or wrong manually, which is something people do when they are practicing queries. So, if the dataset has a smaller amount of data it will be better for teaching.