Project 2: RMDBs

Project Design Document

Team 15

Albin Kyle Myscich

Bryceson Laing

Christian Loth

Rebecca McFadden

Department of Computer Science and Engineering

Texas A&M University

Date

17 February 2020

GitHub:

https://github.tamu.edu/amyscich/RMDB_CSCE315.git

git@github.tamu.edu:amyscich/RMDB_CSCE315.git

Table of Contents

1. Executive Summary

This paper discusses the methods and ideas we hope to implement in our product for the A&M Health and Kinesiology Department. We plan to deliver a database system with a graphical user interface to allow undergraduates to analyze current football data for their introductory sports statistics class. In the following sections, we detail our project's purpose and the need it fulfills, as well as provide information about the design. As per the product specifications, we have discussed the high-level and low-level design for our product. We have enumerated the various entities within our database, their purpose, the risks associated, and their interactions with one another. The data within our database is broken down to a granularity we feel is appropriate to satisfy the queries we believe students will be interested in. Without full knowledge of the data, it is difficult to decide what to include or not, but we are comfortable with the overall concept of our design. In addition, we have commented on the overall benefits, risks, and issues with implementing a system such as this.

2. Purpose of Project

The department of Health and Kinesiology at Texas A&M would like to have students in its undergraduate Sports Statistics class analyze current college football data. Given the sheer number of conferences, teams, and players in the NCAA, the dataset in question is rather large. As such, it would be difficult to analyze this data in a traditional data processing program such as Google Sheets or Excel. The department would like their undergraduates to be able to easily and quickly query the data to answer trivia-like questions pertaining to college football statistics. With so many parameters of a college football player that need to be documented such as touchdowns, field goals, GPA, etc, a SQL data table would be a great model for doing so. All of the data about a player is associated with that player's 'Player ID'.

3. Needs Statement

The current format of the dataset is not suitable for the students in the Sports Statistics class to work with. Therefore, the Department of Health and Kinesiology at Texas A&M has contracted us to design a system to enable the students to efficiently handle and analyze the data. The solution should be straightforward to use and allow the user to query any specific data they request based on different parameters or relationships. For example, a user should be able to pull all the information about a specific player, request the players and coaches for a team, look up the player with the most touchdowns, etc.

4. High-Level Entity Design

Our system would provide a graphic user interface (GUI) that is connected to a database containing the college football data. The GUI will allow the students to easily query the data in a user-friendly way. The database will be set up to allow for efficient calculations of player, team, and conference level statistics.

The high-level entities included in the database are *Conferences*, *Teams*, *Players*, *Individual Games*, *Coaches*, *Schools*, *Statistics*, *Touchdowns*, *Field Goals*, *Referees*, *Tackles*, and *Flags*. The *Conferences* table stores a 'Conference ID' so that a user can look for players and other information within a specific conference. The *Teams* table stores information about the number of players, the ranking of the team in its conference, the school it belongs to, and its location. The *Players* table stores information about the individual players, such as height, weight, age, GPA, etc. The *Individual Games* table contains information about each game played, including the 'Team IDs', score, location, referee, and the winner. The *Coaches* table tracks the pay, start date, age, and gender information about each coach. The *Schools* table contains information about each school, like its size, address, and name.

Statistics is a table of relevant data about a player's performance such as touchdowns, yards run, tackles, and other information. This table is separate from the *Players* table to isolate the player's statistical record from their personal information for more relevant querying. *Touchdowns*, *Field Goals*, *Tackles*, and *Flags* are tables designed to store information about individual events that may occur during a game and the player/team that was involved. The *Referees* table is a table (similar to *Coaches*) to store information about the referee that is officiating each game.

5. Low-Level Entity Design

5.1. Usage

The *Conferences* table is likely the "highest level" table. In terms of "inheritance", *Conferences* contain *Teams*, which contain *Players*. Therefore the *Conference* table has a one-to-many relationship with *Teams* and, by extension, *Players*. By separating *Teams* and *Players* into *Conferences*, the end-users of the product can see data about players from specific regions of the country or compare across two conferences. The risk associated with this entity is that, since we have defined players as belonging to a team and then a conference, we do not account for players moving between conferences over the course of two seasons.

The *Teams* table is just under the *Conferences* table. It stores information about each team within a conference, as well as information about the location of the team and its school and coach(es). As with the *Conferences* table, it could be useful to see data about players from specific teams and compare them. Grouping players by team can allow for interesting views of the data. Again, the risk is that we have not accounted for what happens if a player were to move between teams over the course of two seasons.

The *Players* table just stores a 'Player ID'. This table mostly serves as a connection between the 'Player ID' and the many tables containing data about each individual player. This is useful in our design because it allows us to find data in lots of tables that pertain to a specific player. For example, we can allow the user to find all the touchdowns that any given player made or all the games they played in. However, this complexity can be risky; if tables are requested in a dynamically changing architecture, obfuscated dependencies can lead to potential issues. This can ultimately mangle the overall stability of the database inputs and requests.

The *Schools* table stores the location and population of a school. This information is useful for determining the division a school plays in, as well as game locations. We are able to link teams to their schools and associate individual games to the stadium they are played at. The end-user may also want to see how larger schools compare to smaller schools in terms of statistics. The risk of this is, again, complexity.

The *Statistics* table stores performance information about a player such as the number of touchdowns, number of field goals, etc. This can be used for finding record-holders within a team or conference. An end-user could ask for the total number of touchdowns a team has made. Since we have collected this data for each player, finding the total is as simple as summing the touchdowns per player across the team. Again, the biggest risk is complexity and data duplication.

The *Records* table stores data related to the highlights of each team. The 'Records ID' maintains the table's integrity with respect to uniquely defined records. This information is particularly useful when considering the relationship between teams and their definitive records. Notably, the team's ID provides additional metadata transitively associated with their players. Here, the user can request unique records from within each team. The end-user can use this information to compare records between teams. Ultimately, the risk is embedded in the complexity of the primary and foreign keys.

The *Player Info* table is accessed through the 'Player ID'. It stores basic information about each player such as height, weight, gender, GPA, etc. All of these are filters that can be used to create more views of the data to answer our end-users' queries. The players' positions, team, and number are also stored here. This table serves as the bridge between players and their teams. The risk here is that the bridge may be further from the 'Player ID' primary key and this may overcomplicate the process of locating a player on any given team. However, it does allow for easier filtering of player data based on the 'Team ID'.

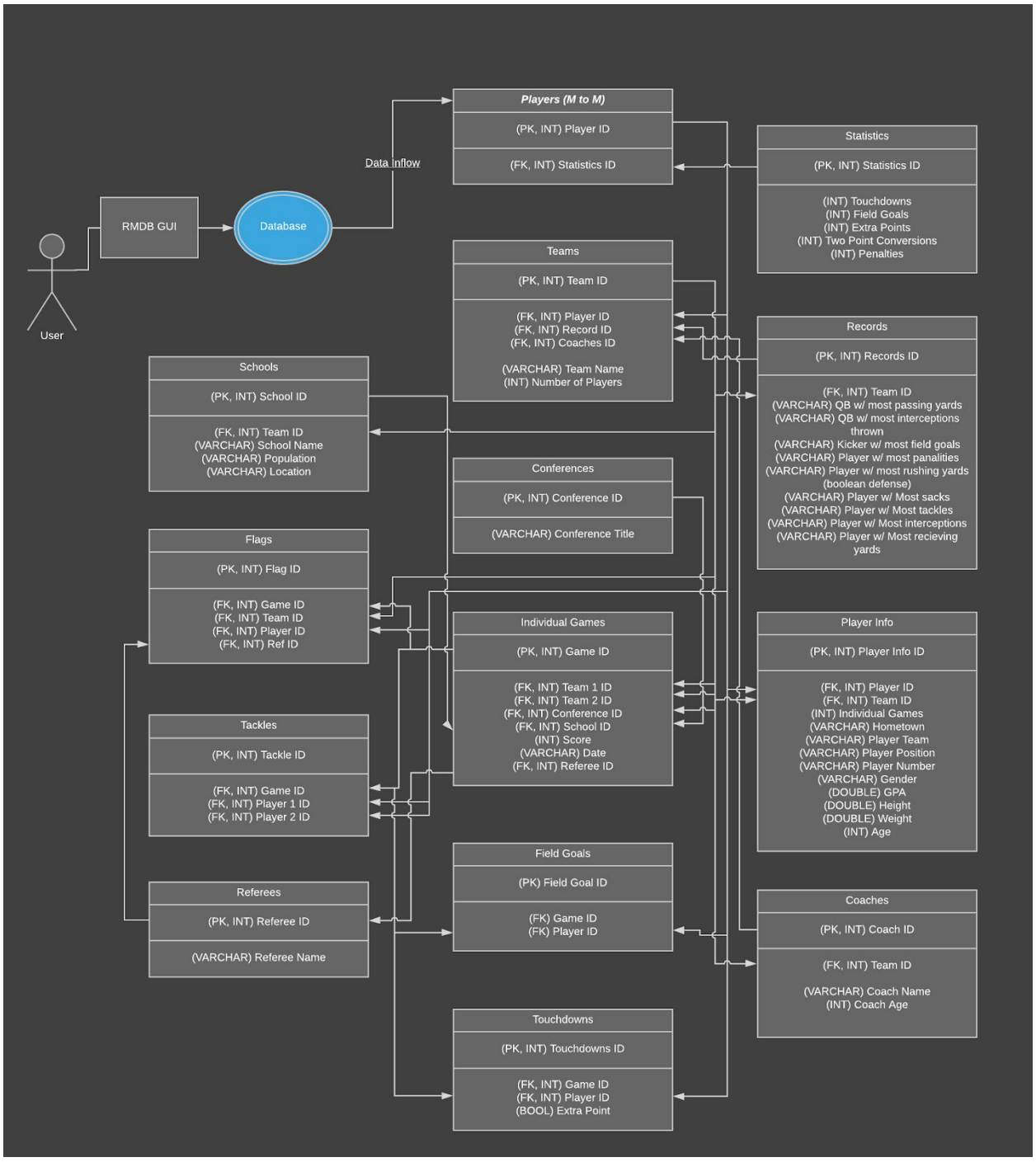
The *Games* table stores information about each individual game such as score, on duty referee, which teams played, the location of the game, and the date. By removing games played from the *Teams* table, we allow both the home team and the away team to have a one-to-many relationship with the games while eliminating some possible data overlap. Again the risk here is complexity.

The *Touchdowns* table tracks individual touchdown events that can occur during a game. It stores the 'Player ID' who made the touchdown and the game during which it occurred. This table, in addition to the *Tackles, Field Goals,* and *Flags* tables are designed to track individual scoring/penalty events and the parties involved. Removing these from the game table and making them separate events allows for a one-to-many relationship and allows the end-user to filter these events by player or other attributes.

The *Referees* table holds information with respect to the 'Referee ID', their name, and the games they reffed (based on the 'Game ID'). As previously mentioned, the *Referees* table allows the user to select metadata from a game and is able to use its primary key as a reference from foreign tables. Attributes of risk are associated with minor complexities in table references between both games and flags made during a game.

The *Coaches* table carries information about each team's coach such as their age, name, and pay. The benefit here is the ability to filter team statistics based on the attributes of their coaches. For example, do teams with higher-paid coaches perform better than those without? There is not a lot of risk here, as coaches are only tied to their team, but it does add complexity to the system, especially if a coach moves teams.

5.2.1 Model



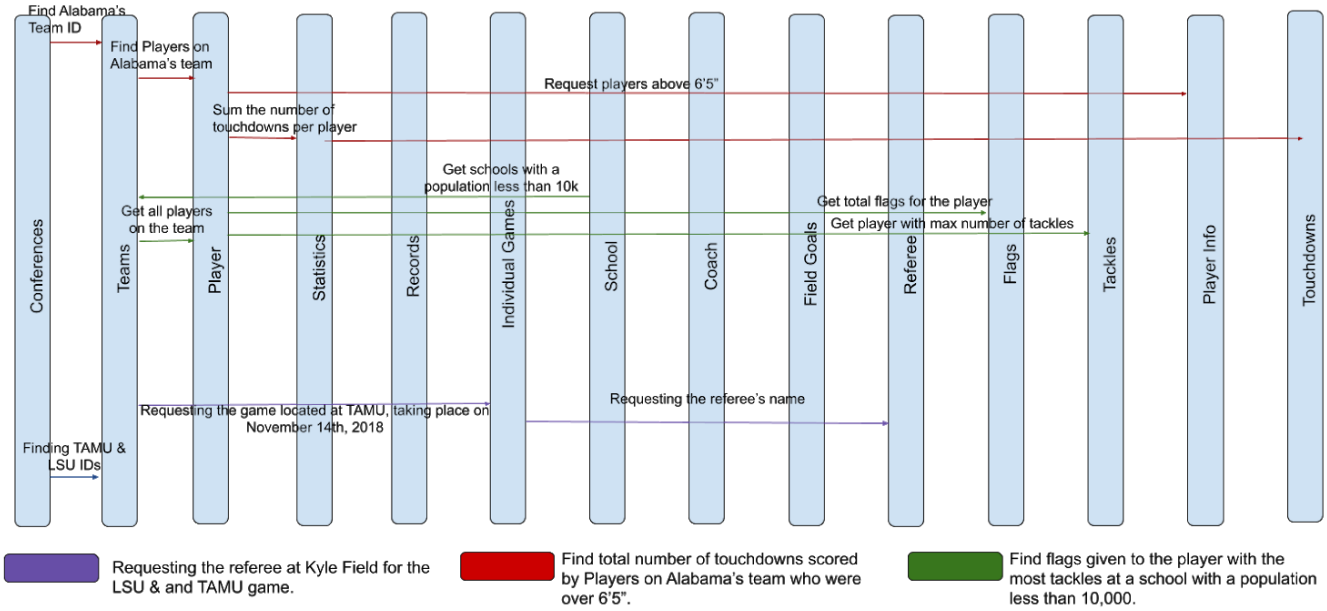
5.2.2 Model Description:

Beginning with the user's interactions with the RMDB (Relational Model Database), the GUI will provide a streamlined method of inserting and removing data. Of course, the manipulated data must be passed into the database and organized or removed in their respective tables where it's parsed. The *Conferences* table is used to categorize teams and players. It has a one-to-many relationship with *Teams.* The *Teams* table is used to categorize individual players. The *Players* table will act as an index to access the *Statistics* and *Records* for each individual player. The *Conferences* table has a one-to-many relationship with *Teams*, and *Teams* has a one-to-many relationship with *Players*. The 'Conference ID' primary key is used as a correlative reference for both

the teams and players within that conference. To clarify, there can only be one primary key per table in SQL, but in our case, we are the same keys across tables. For example, each player's personal data is stored in the Player Info table, while their performance in games overall is stored in the Statistics table, and the rows share the same 'Player ID'.

Teams and *Players* also have a one-to-many relationship with *Games*, and therefore also with scoring mechanisms such as *Tackles*, *Touchdowns*, *Field Goals*, and *Flags*. These are used to track individual events within games and as such, they are connected to games with the 'Game ID' and 'Player IDs' of those involved. *Referees* are also connected to the games they reffed. The *Coaches* are not connected to individual games, but rather to their respective team (and thereby to the school). The *Schools* are connected to each game to provide pertinent information about where each game took place.

5.3.1 Interaction Diagram



5.3.2 Description:

In the first interaction we detailed, we found a particular referee by initially requesting 'Team IDs' (in our case, LSU and TAMU) in order to get the teams' names. From here, we conditionally find the specific game using the November 14th, 2018 date in addition to the team names. Finally, the referee's name is requested using the foreign 'Referee ID' key, which is found in the *Referees* table.

The second interaction we detailed was finding the total number of touchdowns scored by players on Alabama's team who are over 6'5". First, we find the 'Team ID' for a team in the SEC with the name "Alabama" and get all the players on that team. Then we select the players whose *Player Info* says they are greater than 6'5" in height. Finally, we sum their total touchdowns by getting those values from the *Statistics* table using the 'Player ID'.

The third interaction was to calculate the number of flags given to the player with the most tackles at a school with a population of under 10,000 students. The *Schools* table can be queried to find all schools with a population under 10,000 students and get the 'Team IDs' related to each of those schools. From there, we can find all players at each of those schools and use the 'Player IDs' to get the total number of tackles for each player from the *Statistics* table. Once the player with the maximum number of tackles is found, we can find their total flag count from the *Statistics* table.

6 Expectations

6.1 Benefits

The database is constructed in a way such that closely related data is split up into relatively small pieces. This will allow the user to easily query specific data without receiving unnecessary information. This should increase the speed of data fetching and decrease the amount of data sent to the client. It will also allow for more filtering of data based on criteria. This will enable students to get the answers to very specific trivia-type questions that they are asked to answer for their classwork.

The design is also very relational. High-level entities are nested within each other, e.g. 'Team IDs' are stored within the player entities. This lets the user quickly fetch the team that the player is on based on information within the player entity and vice versa. This also lends itself well to the A&M Department of Health and Kinesiology's desire to have their students answer trivia-like questions, as the students will be able to quickly find specific statistics about players, teams, or conferences.

In addition, as our end-user will interact only with the graphical user interface (GUI), they don't need an in-depth understanding of databases, SQL, or even what relations are occurring behind the scenes. This complexity is abstracted from our user. This way, they can focus on their classwork and understanding the relevant material and not be hampered by the task of learning data manipulation.

6.2 Assumptions

We are assuming that the data is just over the period of one year, and therefore players, cannot be on multiple teams in the same year. We are also assuming that the end-user is interested in all of the statistics for each type of player, as we did not break statistics into separate categories for running backs versus quarterbacks versus kickers, etc.

We have also assumed that our GUI will be structured in a way to allow users to filter data on the *Conferences*, *Teams*, and then *Players* level. We made assumptions that they might be interested in narrowing their focus to individual games or events during a game. The data may not be provided at this granularity, but we have assumed that we will get data that will allow us to offer these narrow view frames.

6.3 Risks

We don't know the details of the data we will be given, so we are risking the entire design of the database because it might not be suitable for the final application depending on how the data needs to be structured. If the data type does not correspond to the data type we have in our table (ex. a string instead of an int), we risk having to spend time to redesign it. We have also made assumptions about the data tables we will be getting and the granularity of those tables, which could need redesigning if we don't end up getting data about things (such as individual touchdowns/field goals or coach information). Besides this, we have not created a design for our end-user GUI yet, so we are unsure of all of the possible interactions our data tables may need to have. This may cause us to have to redesign certain parts of our interaction and UML diagrams.

We have also designed a database made up of a significant number of small tables. This could create unnecessary complexity and introduces the possibility of data duplication. We have a lot of information that is separated and accessed using the 'Player ID' or 'Team ID', which could create confusion when we go to build the final product with the GUI.

6.4 Issues

It doesn't take into consideration the fact that there are multiple seasons, so players might be on different teams for different seasons. With this, there is the risk that if a player does switch teams that his statistics will go over to the other team, and be invalid. We are unsure if this will be an issue in the final design, as the specifications did state that we were dealing with current football data. We are aware of the possibility of this issue.