Design Document

==========

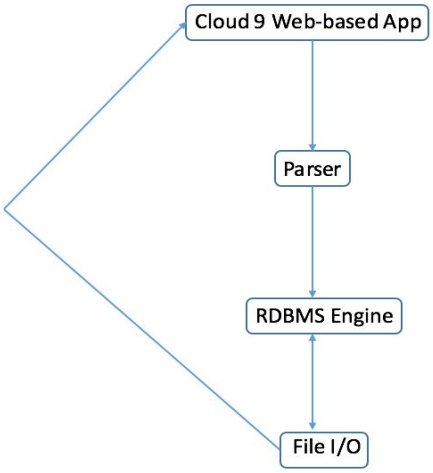
Purpose of the Project:

The purpose of this project is to create and implement a generic relational database management system. This database is designed as a sports management system. It will store information based on the sport, the player, the team, and a record of the team's wins, losses, and ties. The player's given attributes include, but are not limited to: name, age, points scored, and jersey number. The team's attributes include, but are not limited to: team name, wins, and losses. Lastly, the sport's attributes include, but are not limited to: sport name, number of players on the field, and playing surface.

High Level Entities

The picture below represents the Entity-Relationship diagram. This is simply a picture of how the information in our database will all work together.

Below, we have the visual representation of the High Level Entities.



The process starts with the interactive system when the user decides to input data. From there, the parser will accept the pseudo-SQL commands generated by the interactive system. The parser will then parse those commands and pass the java equivalent to the engine. The engine will then determine if the java commands are valid and execute accordingly. Lastly, the engine passes all of the data to the json file where we will keep a record.

Engine

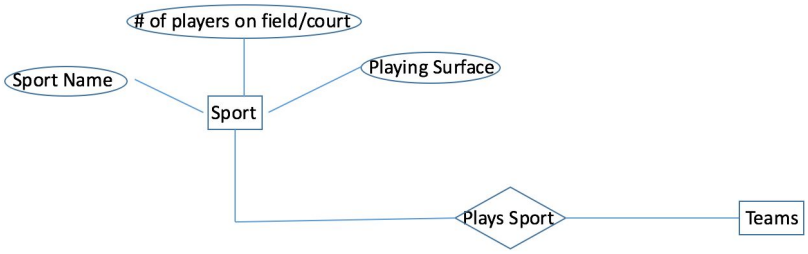
Purpose:

The engine is used to do all the dirty work. This is where the bulk of all operations are done from. If the input is valid from the interactive system, the parser will send the translated input to the engine to be executed. If the input is not valid, the parser will throw an error. Furthermore, the engine will not work directly alongside the interactive system. Meaning that, the engine will work in between the json file and the parser. The engine does all of the operations (insert, delete, select, etc.).

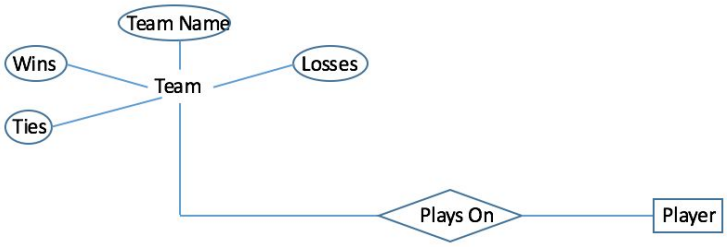
Low Level Design:

The low level design will consist of the three objects: Player, Team, and Sport. Each entity will have a corresponding class file to be used within the engine. Inside each class file, the given attributes for each entity will be declared as members. This approach will help to simplify the code as well as increase readability. The attributes for each entity are shown below:

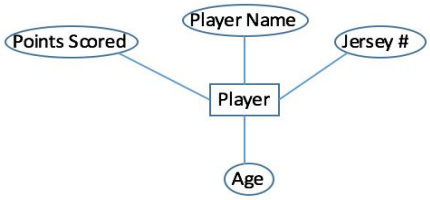
- Sport:
    - Name
    - Number of players on field
    - Playing surface (gym, field, etc.)



- Teams:
    - Team name
    - Location
    - Wins
    - Losses
    - Ties



- Player:
    - Name
    - Age
    - Jersey number
    - Points scored



The main data structures in this database are hashmaps. We decided to go with hashmaps because the <key, value> setup allows for easy identification of what values are actually stored within the structure. Furthermore, hashmaps allow for quick insert/delete/lookup function calls at just O(1). This is crucial for a database considering that managing data revolves around the

ability to insert, delete, and lookup data. The process of giving each entity its own key will be related to combining two static attributes to create a unique ID for each object.

With all that being said, the three entity requirement has been met. Furthermore, each entity will have its own table. Each of our two relations will also be represented in table, also. This means that we will have met the five table requirement, as well.

Benefits/Assumptions/Risks/Issues:

The benefits of this approach rest in the simplistic design. Although the design relies heavily on the engine, creating player, team, and sport objects will help to organize the code and make it readable. Furthermore, as said before, the time complexity of the hashmap functions allow for quick operation. The main assumption that is being made is that the time complexity of the given hashmap functions operate under O(1). Under the worst circumstances, it could take up to O(n). However, the average case states otherwise. One of the risks of this implementation include the heavy reliance on the engine. As mentioned earlier, the engine is doing all of the dirty work while the parser and interactive system simply work in accordance with it.

Parser

Purpose:

The parser is used to interpret input (pseudo-SQL commands) from the user and call correlating functions from the engine. In essence, the parser's primary objective is to translate from SQL to java. The output is then brought back to the engine to await execution.

Low Level Design:

The parser will be implemented through the use of a recursive descent parser. This is because the SQL can be nested. For example, SELECT * FROM (SELECT * FROM <table>) is a valid SQL call. That is why the parser must be recursive.

Benefits/Assumptions/Risks/Issues:

Using a recursive descent parser helps to simplify the code. If done right, this will eliminate all of the tedious, time-consuming implementation of parsing each command individually. However, recursive parsers tend to be hard to implement. One of the biggest risks/issues is the lapse in understanding that can sometimes come with recursion.

Interactive System

Purpose:
We have two plans for the interactive system, Plan A and a backup Plan B. Plan A is to go for the extra credit and host the interactive system online, using HTML and JavaScript to render the UI. Plan B is to simply use the terminal and have the user type out data and operations. Despite these differences, the overall idea is relatively the same - a Sports Management application.

The interactive system will serve as the bridge between the common man and our program. It will present itself as a Sports Management application, and we hope to be able to publish it to a free hosting service that can be accessed on any browser. If that proves too difficult, we will revert to simply using a terminal-based application. If we are successful with the website, however, all the user will have to do is input the URL. From then, the user see several tables, with perhaps arrows between them to represent relations, and the user will have the option to either add new data or manipulate current data. As far as the Terminal application goes, it would simply consist of the user typing in data in a directed format (The user will not have to be familiar with any SQL grammar).

Low Level Design:
If time permits, it will be hosted on https://c9.io/, a free website hosting service. We will use HTML to create a simple canvas, and then JavaScript to render the UI that the user will interact with. The UI will primarily consist of several tables, representing the different entities and their data. There will also be a menu with a hierarchy of buttons and drop-down menus with predefined inputs. From these, the user will easily be able to input and manipulate data.

The week objectives for this portion of the assignment are as follows:

- Create the HTML index file rendering a canvas
- Use JavaScript to fill the canvas with tables, buttons, and menus that, when interacted with, will send SQL- based statements to be parser via API.

Benefits/Assumptions/Risks/Issues:
One of the hardest challenges to tackle with this will be the API. Since we will be using JavaScript to send the SQL-based grammar to our Java parser, we will need to make sure it is done as efficiently as possible. If each SQL operation is represented as a button on the canvas, we will be able to then grab data from text boxes and send that to the parser. However, finding a way to aesthetically do this might be difficult.

I think in the long run, an easy-to-access website will be extremely beneficial to the end user, and since JavaScript isn't that difficult to handle, we should be fine. We just need to manage our time wisely and make sure the parser and engine function as designed here.