

Introduction

My application is called Matchup League. Matchup is a card game I created when I was younger, in which a team of 'fighters' faces off against another team of fighters. Each individual fighter is lined up against a fighter on the other team, called a match. If one fighter's strength is one of its opponent's types, the strength bonus is added to their base power. The same happens with weakness, but is subtracted. The fighter with the higher base power after applying strengths and weaknesses wins the match, and the team that wins more matches wins the game.

I chose to make my application with this game because it's something I've been working on and improving for many years. Until this project. I've had to do all the organizing for Matchup League by hand and on paper, which can become very time-consuming. By using a database to store information, I can make calculations and visualize data faster than I ever could on paper.

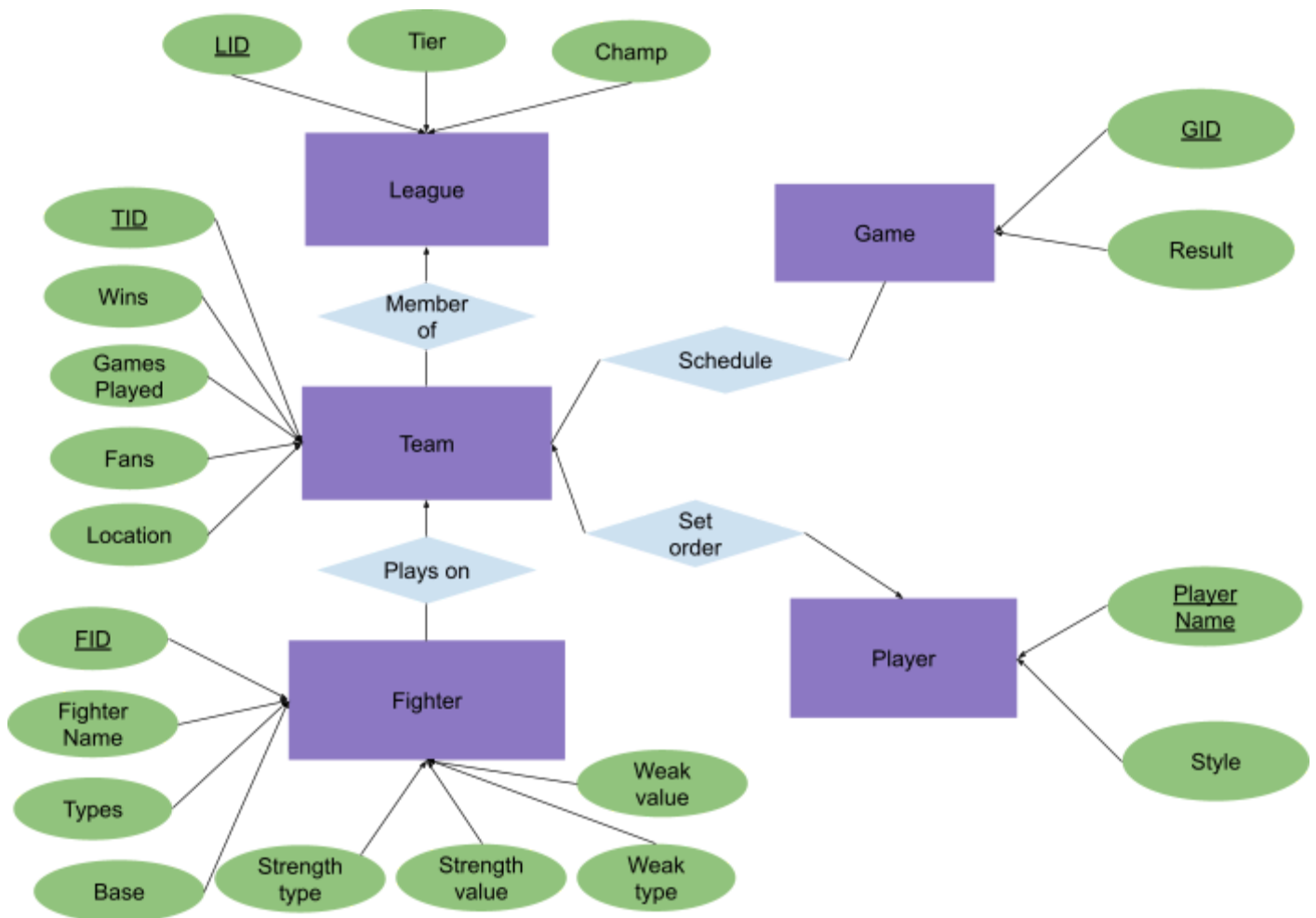
Database details

The application is organized as such: 7 fighters make up a team, 4-16 teams make up a league, and there can be multiple leagues. Each team has one player, whose 'style' decides how the team is managed or if a real user is controlling the team. All the teams in a league play each other once, and the games are generated automatically and stored as a schedule.

The data for fighters was made by me, based on several shows/movies I like, as well as other various sources, and a few of my own creation. The sample team file includes two leagues, Florida and Minnesota, whose teams are the major population centers in their state.

Origin of all fighters in sample file:

- Neon Genesis Evangelion
- Jujutsu Kaisen
- Fullmetal Alchemist
- Avatar: the Last Airbender
- Public domain
- North American folklore
- All other fighters are original



ER Model

Here's a description of a few of the columns:

- **Team fans:** An int that represents the size of that team's hypothetical fanbase. The amount of fans was determined by filling out a map, making a region for each team, and dividing the resulting population in each region by 10,000 (for example a region with a million population will have 100 fans). A bye team will have -1 fans.
- **Team location:** For location-based leagues, denotes intended league to be added to.
- **League tier:** A future feature will be promotion/relegation, and the tier is a char from A to D that determines where in the ladder a league is.
- **Player style:** An int that denotes a player's play style. 0-3 are CPU styles, and 99 denotes a real user. -1 denotes a player associated with a bye.

Relational schema	Dependencies
Fighters(<u>FighterID</u> , FighterName, Types, Base power, Strength type, Strength value, Weakness type, Weakness value, <i>TeamID</i>)	FighterID -> FighterName, Types, Base power, Strength type, Strength value, Weakness type, Weakness value, TeamID
Teams(<u>TeamID</u> , wins, gamesPlayed, <i>Fighters</i> , <i>LeagueID</i> , <i>PlayerName</i> , <i>GameSchedule</i>)	TeamID -> TeamName, GameSchedule, LeagueID, Fighters GameSchedule -> wins, gamesPlayed
Leagues(<u>LeagueName</u> , Champion, Tier, <i>Teams</i>)	LeagueName -> Tier, Teams Teams -> Champion
Games(<u>GameID</u> , <i>Teams</i> , Result)	GameID -> Teams Teams -> Result
Players(<u>PlayerName</u> , Style, <i>TeamName</i>)	PlayerName -> Style, TeamName

All the relations are in BCNF, as all variables can be determined from their primary key.

Functionality details

- Load data: Currently, the program is hard-coded to read in specific csv files to insert into the database. These files can easily be edited or added on to before running the program.
- Add user: The program user has the choice to either simulate all teams or to take control of a team. A user in charge of a team can draft their own fighters and choose the order in which to play them.
- Fighter survey: Gives helpful stats like the frequency of each type, average base power, and average strength/weakness by type.
- Autogen: The advanced function of the project, Autogen is a feature where the user can insert any number of randomly generated tuples into the database. The user enters a number of desired teams, and the program will auto-generate teams, fighters, leagues, and players accordingly. These autogen tuples are given a unique UUID (8-character hexadecimal) as their name, and random stats within reasonable parameters.

- Play season: The playable game consists of three phases:
 - Fighter draft: Teams will take turns selecting fighters from the pool of undrafted fighters. The draft order is determined by the team's number of fans. If the user chose to control a team, they can decide the fighter to draft. The program displays the top three choices, but the user can select any free fighter.
 - Schedule generation: The game schedule is automatically generated via a round-robin format, meaning each team in a league plays every other team in the league exactly once. If a league has an odd number of teams, a bye team is added, which cannot play games nor have fighters but takes up a place in the database to balance out the schedule. Bye teams are denoted with -1 fans and -1 games played.
 - Game rounds: Where the game of Matchup actually happens. Each team plays one game each round (excluding byes) and the results are recorded in the database and displayed. At the end of each round, each league's standings are displayed. After all rounds, each league's champion is announced.

Implementation details

The program is written in Java, using Hibernate API to connect the code to a MySQL database using a local connection. Tables are constructed by annotating Java classes as entities, and SQL queries can be run from Java classes. Due to time constraints, there is no front-end interface but the program is still fully usable and readable using the console output. The database can also be accessed via MySQL.

Here are the non-entity classes and their function:

DAO<T>	Data Access Object: A template class, which executes queries. Serves as the main communication between the database and the program. Can commit transactions on mutation queries.
Manager	Handles the assigning, unassigning, and transfer of tuples to their foreign relations. By extension, handles the draft, scheduling, gameplay, and auto-generation. Can commit transactions.
Main	The driver of the program, holds the menu so one can use the program

Schedule	Generates games for each league. The program makes an implicit table called schedule for the many-many relationship between Team and Game, but it is not explicitly implemented in this class.
Entities	An abstract class which each table extends. Constructs a DAO class for each table in order to access and modify data from them.

Below are a few example queries that are carried out in the program:

Game feature	SQL query
Assigning team to league by location	UPDATE Teams t SET t.LID = (SELECT LID from Leagues lg WHERE lg.name = t.location);
Draft (getting undrafted fighters)	SELECT * FROM Fighters WHERE team IS NULL;
Draft (getting teams ordered by fans)	SELECT * FROM Teams ORDER BY fans DESC;
Getting a round of games for a league	SELECT DISTINCT g FROM Games g JOIN g.teams t JOIN t.league lg WHERE g.round = (round) AND lg.name = (lg.name);
Determining standings (given that more than one game has been played)	SELECT t.name, t.wins, t.gamesPlayed - t.wins FROM Teams t JOIN Leagues lg ON t.LID = lg.LID WHERE lg.name = (league name) ORDER BY wins / gamesPlayed;
Fighter survey	(for each type) SELECT count(*) / (SELECT count(*) FROM FIGHTERS) FROM Fighters WHERE Types LIKE '%(type)%';


	SELECT count(*) / (SELECT count(*) FROM FIGHTERS), avg(strVal) FROM Fighters GROUP BY strType; SELECT count(*) / (SELECT count(*) FROM FIGHTERS), avg(wkVal) FROM Fighters GROUP BY wkType;
--	--

Experiences

From working on this project, I learned that a program can become very complicated very quickly if you are not careful. I also learned that it's very important to keep the flow of control and the flow of data as centralized and restricted as possible. Databases with multiple tables all interacting with each other takes careful planning and extensive testing in order to work as intended. In the future, there a lot more features I'd like to implement: different tiers of leagues, where if you win your league you are promoted to a higher one; pictures for each fighter/team; a customizable home stadium; individual fighter stats; a nice-looking Web interface; and the ultimate goal which is online functionality.

References

Program on Github: <https://github.com/lognog2/Matchup-League>

Matchup explanation slideshow:  Matchup explained

Database organization:  Databases

Fan maps: [Florida Map](#) [Minnesota Map](#)