J5 Pocket Monsters Database

Jacob Janzen
janzenj2@myumanitoba.ca

Daniel La Rocque
larocq17@myumanitoba.ca

Jared Webber
webberj1@myumanitoba.ca

Colin Johnson
johns233@myumanitoba.ca

December 10, 2021

1 Data Summary

Pokémon was fresh on our minds when we began the project because it had been mentioned in a class we have together several times during it. We very quickly realized that a database containing information about Pokémon would be a great way to have lots of data with very interesting relationships.

We ultimately decided to constrain our idea to one specific game in the franchise while building our data model because we wanted to track data about non-player characters and locations in the games which would quickly get very complicated when discussing multiple games. It would introduce even more complications when considering the fact that many relationships between entities would change depending on the specific game. To solve this, we chose one specific game rather arbitrarily: Pokémon Emerald.

The Data consists of the set of Pokémon that are present in Pokémon Emerald, the locations, moves, and non-player *trainers* in the game, as well as the *types* that Pokémon and moves can have. We tracked the methods in which each Pokémon can learn each move as well as the location of all non-player characters and the locations where each Pokémon can be found.

In the end, we had a 1.4 MB database. Our largest table, `Learns`, had 15280 records while our smallest table, `HM`, had only 8. In total, there were 24547 records in the database.

2 Data Model

We gathered data based on the data model we created, which was created from brainstorming every type of data we could gather about the game that we deemed interesting and then linked together in every way that we could think of.

Most of the challenges in creating the data model were in fine tuning the details after working out exactly which data we would use. For instance, originally the `Learns` table contained every move that every Pokémon could learn without

any distinction of method. Later, we decided to add the method and split it into one relationship for each method in which a Pokémon could learn a move. This added a lot of unnecessary complications to the data set and made queries far more difficult than they needed to be. Thus, we combined them back into a single relationship with a method attribute being a primary key on the relationship. Unfortunately, learning a move by breeding is a little more complicated than other methods of learning a move. It is dependent not only upon the Pokémon learning the move and the move itself, but also the father of the Pokémon learning the move. Therefore, we separated that ternary relationship between two Pokémon and move from the rest of the learn methods.

Another challenge that came up in the project was the relationship between a trainer and the Pokémon that they used in their team. Originally, we had `Team` as a weak entity that depended on `Trainer` and `TeamMember` as a weak entity that depended on `Team`. This two-layer weak entity added a lot of unnecessary complication, so we changed `TeamMember` to be a relationship instead which has an ID as a part of its primary key.

A tricky participation ratio was surrounding the `HasTypes` relationship. The ??? type which existed in Pokémon Emerald was a type only used for the move *Curse*. Thus, we had only partial participation from `Type` to `Pokemon` and total participation from the other side because every Pokémon has at least one type and can have two.

A tricky cardinality ratio is through the `EvolvesFrom` relationship. While generally, when a Pokémon evolves, it can only evolve into one Pokémon, there are rare situations where one Pokémon can evolve into multiple different ones. For instance, *Eevee* can evolve into *Vaporeon*, *Jolteon*, *Flareon*, *Espeon*, or *Umbreon* depending on the specific circumstances that it evolves under. This relationship does not work in the opposite direction. There are no cases where one Pokémon evolves from two different Pokémon in Pokémon Emerald.

3 Database Summary

Table	Cardinality	Arity
Pokemon	386	17
Abilities	610	2
EggGroups	504	2
Location	106	1
Move	354	6
Trainer	493	3
Type	18	2
FoughtAt	520	2
FoundAt	647	6
HasTypes	557	2
HM	8	2
Learns	15280	3
LearnsByBreeding	2007	3
Team	887	3
TeamMember	1795	6
TM	50	2
Effectiveness	324	3

4 Queries

- The Pokémon in a given egg group.
- The Pokémon a given Pokémon can breed with.
- The locations a Pokémon may be found at and the method which they can be found with.
- The locations a Pokémon of a given type can be found.
- The locations that a Pokémon with two given types can be found.
- The locations with a given trainer.
- The locations where a certain trainer class can be fought.
- The locations where a certain Pokémon of at least a certain level can be found.
- Moves that are learnt by all Pokémon and the method by which they are learnt.
- Moves that are learnt by a given Pokémon.
- Moves that are learnt by a given Pokémon and the method by which they are learnt.

- Moves that have a given effectiveness against a given type.
- Moves that are super-effective against a given Pokémon.
- Moves that are neutral against a given Pokémon.
- Moves that are not-very-effective against a given Pokémon.
- Moves that a given Pokémon is immune to.
- All moves' effectiveness against a given Pokémon.
- List all status moves.
- Ways a given Pokémon can learn a given move.
- Moves of a given type that a given Pokémon can learn and the method by which they are learnt.
- Moves a Pokémon learns through breeding.
- Moves a Pokémon can learn through breeding with a given father.
- All Pokémon hatch times.
- All move names.
- Pokémon and their Pokédex position.
- All trainer data.
- Stats of a given Pokémon.
- Evolutions of a given Pokémon.
- Pokémon with a given move.
- Pokémon from each type with the highest value in a given stat.
- Pokémon from each type with the lowest value in a given stat.
- Pokémon having a minimum value of a given stat.
- Pokémon having a maximum value of a given stat.
- Pokémon which can be caught at a given location.
- Pokémon which can be caught at a given location from a given encounter.
- Pokémon which can learn a move that is super-effective on a given Pokémon.
- Pokémon that can be caught at a given location that can learn a move that is super-effective on a given Pokémon.
- Pokémon that a given move is super-effective against.

- Pokémon that a given move is neutral against.
- Pokémon that a given move is non-very-effective against.
- Pokémon that are immune to a given move.
- Effects a given move has on a Pokémon.
- All abilities Pokémon have.
- Pokémon with a given ability.
- Pokémon of a given type that can learn moves of a given type and the methods by which they are learnt.
- All teams of a given trainer.
- All teams with a given Pokémon.
- All team with a minimum level.
- All teams with a maximum level.
- Number of Pokémon per type.
- Types with physical damage.
- Types with special damage.
- All TM moves
- All HM moves
- All moves all Pokémon learn by TM
- All moves all Pokémon learn by HM
- All moves a given Pokémon learns by TM
- All moves a given Pokémon learns by HM

5 Interface

The project was created with an Electron front-end written in JavaScript with Vue.js. The back-end is a Flask application written in Python. The back-end runs the queries which are requested through API calls from the front-end using Axios as an HTTP client. The application can be started with the following on Linux/macOS:

```
.../3380_Project $ ./run-linux.sh
```

and on Windows:

```
...\3380_Project>./run-windows.bat
```

On both, you can then visit `http://localhost:8080/` to see the application. To close the application, press CTRL+C to close the front-end and then again to close the back-end.

The interface is fairly simple. There are four tabs at the top of the screen. Starting from left to right, they are: a homepage containing information about us, a link to our repository, and details about how to navigate through the application; a copy of this report; our Enhanced Entity Relation Diagram; and finally, the database interaction page.

Interacting with the database is quite simple. You can start by clicking the drop-down menu saying "What do you want to know?" Afterwards, you can start typing to search for a specific query or to filter down the list, or just scroll down the list of queries until you find your desired query. You can also use the checkboxes to narrow the search down to specific types of queries. After selecting a query, it may open up new drop-down menus to select specific parameters of the query. When all the fields have been selected, you can click "QUERY!" to execute the query and the results will appear on the page. If you want to save a CSV file containing the results of the query, you can click download. To execute another query, click "NEW QUERY."