Project : Pokédex

Emily Greiman, Garrett Quintero, Vishaal Sridhar

CSCI 403 : Database Management

Part One A : The Dataset

We decided to choose our dataset based on the first section of interesting information we could find. We found a Pokemon database from Serebii.net. Serebii is one of the largest collections of information about the pokemon franchise available online, and we got our data from here :

https://www.kaggle.com/rounakbanik/pokemon/data

Once we found our nerdy data, we decided that we wanted to figure out something to do with it. It's pokemon, so why not, right? Since we had all of this data about pokemon from several generations of the games, we decided to build a mini search engine with the characteristics that we thought might be vaguely relevant. Thankfully, the data was published under the CC0: Public Domain license, so we didn't really need to do anything special to use the data.

Part One B : The Table

The dataset we were pulling from had 41 columns to choose from, and we wound up collecting the data from 35 of them. The columns that we left out where things that we decided people really wouldn't want to use to search for pokemon, like their catch rate, base happiness, and proportion of males to female.

When we went to load the data into the database, we decided that we wanted to build a custom importer so that it'd be in the format we wanted and only have the columns we wanted. While we were creating the importer, we wound up creating two supporting classes as well, Pokemon and PokeBaseManager. We even got carried away and taught ourselves how to make a

```
String start = "DROP TABLE IF EXISTS pokemon; "
        + "CREATE TABLE pokemon ("
        + "abilities varchar(100), "
        + "against_bug DECIMAL(2,1), "
        + "against_dark DECIMAL(2,1), "
        + "against_dragon DECIMAL(2,1), "
        + "against_electric DECIMAL(2,1), "
        + "against_fairy DECIMAL(2,1), "
        + "against_fight DECIMAL(2,1), "
        + "against_fire DECIMAL(2,1), "
        + "against_flying DECIMAL(2,1) , "
        + "against_ghost DECIMAL(2,1), "
        + "against_grass DECIMAL(2,1), "
        + "against_ground DECIMAL(2,1), "
        + "against_ice DECIMAL(2,1), "
        + "against_normal DECIMAL(2,1), "
        + "against_poison DECIMAL(2,1), "
        + "against_psychic DECIMAL(2,1), "
        + "against_rock DECIMAL(2,1), "
        + "against_steel DECIMAL(2,1), "
        + "against_water DECIMAL(2,1), "
        + "attack INTEGER, "
        + "classification varchar(100), "
        + "defense INTEGER, "
        + "height DECIMAL(5,2), "
        + "hp INTEGER, "
        + "name varchar(50), "
        + "number INTEGER PRIMARY KEY, "
        + "sp_attack INTEGER, "
        + "sp_defense INTEGER, "
        + "speed INTEGER, "
        + "type_one varchar(50), "
        + "type_two varchar(50), "
        + "weight DECIMAL (7,2), "
        + "generation INTEGER, "
        + "legendary BOOLEAN "
        + ");";
```

builder class for our Pokemon class. We originally tried to create the table by having all of our columns listed in an array, but we kept running into syntax errors until we just wound up using the string in the wonderful picture above. Luckily, all of the Pokemon ever created have been given an official pokedex number that we can easily use as our primary key and main identifier in our table. For our sanity, we set up our program so that it would create the database in the `flowers.mines.edu/csci403` database. Each of us would log in with our own credentials and the program would create our own copy of it.

Part Two : Project : Pokédex

Once we had the data loaded into a database, we started setting up ways to search the pokemon in the database. We eventually implemented ways to look up all the pokemon at once with their ids, to look them up ten at a time, to get all of the information of a pokemon based on its id, to look through based on their names, to look them up based on type, to look them up based on which game generation they're from. The type and generation searches return tables of pokemon to read through so the user can select which pokemon they want. We've also been thinking about creating a graphical interface to make it easier to view all of the results and creating a method of combining searches. Below is a screencap of our console interface for looking up the wonderful Jigglypuff.

```
Select a pokemon (0-9). Press '[' or ']' to scroll. Press 'q' to quit
[1]     Nidoqueen
[2]     NidoranM
[3]     Nidorino
[4]     Nidoking
[5]     Clefairy
[6]     Clefable
[7]     Vulpix
[8]     Ninetales
[9]     Jigglypuff
[0]     Wigglytuff
9


--------------------------------------------------------------------------------
Dex-Entry #39
Jigglypuff the Balloon PokÃ©mon
The normal and fairy type Pokemon
Height: 0.5 Weight: 5.5
First appearnce in Generation 1

Abilites: *Cute Charm*  *Competitive*  *Friend Guard*

Damage Taken Modifer against type
Water   |fire   |Grass  |Bug    |Dark   |Dragon |electric   |fairy  |fight  |flying |ghost  |ground |ice    |normal |poison |psychic    |rock   |steel
-------------------------------------------------------------------------------------------------------------------------------------------------------
1.0 |1.0    |1.0    |0.5    |0.5    |0.0    |1.0        |1.0    |1.0    |1.0    |0.0    |1.0    |1.0    |1.0    |2.0    |1.0        |1.0    |2.0

---Base Stats---
HP: 115     Speed: 20
Attack: 45  Special Attack: 45
Defense: 20 Special Defense: 25

This Pokemon is not Legendary.

Press 'c' to continue
```

Part Three : The Specialty Code

While we were trying to get our data ready to process, we wanted to make a few classes to make it easier to add each and every pokemon into the database. The two more important classes we created, PokeImporter and PokeBaseManager, were specifically created to import all of the pokemon into a table and to be the tool we use to look up information about the pokemon. With our PokeImporter, it was created to that we could put one line of code into our main file to load all of the pokemon from a .csv file. With that class, we converted what would have been 100 lines of code into:

```
new PokeImporter("src/pokemon.csv").import_pokemon();
```

The sole purpose of the PokeImporter class is for this single line of code. We wanted to simplify the overall code inside of our main class. This class was supported by the PokeBaseManager class which just provided some of the supporting functions that interfaced with our database. This class also provides the connection that we work with in our main function.

Part Four : The Challenges

There were a few challenges we ran into while we were putting together our project. One of the largest was the formatting of the SQL strings that we were trying to use. Often times, we would have been mistyping a column name or

misplacing a comma. We eventually got around this during the importing of the pokemon by using the pokemon class's data and saving our column names in an array to keep us from misspelling them. Once we got past most of the errors that we were having with formatting, we just had one pokemon that just refused to let itself be imported: Farfetch'd. We remembered there being a scrubber available for SQL statements that uses escape characters for cases like this, but we felt like it was a little much to implement for what we knew would be one case that caused this problem. Instead we just added the proper escape character ourselves.

```
private String formatPokemon(String string, Pokemon pokemon) {

    String abils = "'";
    for (String abil : pokemon.getAbilities()) { abils = abils + abil + " "; }
    string = string + abils + "', ";

    /** add in the against values */
    for (Double against : pokemon.getAgainst()) { string = string + Double.toString(against) + ", "; }

    // ATTACK
    string = string + Integer.toString(pokemon.getAttack()) + ", ";
    // CLASSIFICATION
    string = string + "'" + pokemon.getClassification() + "', ";
    // DEFENSE
    string = string + Integer.toString(pokemon.getDefense()) + ", ";
    // HEIGHT
    string = string + Double.toString(pokemon.getHeight()) + ", ";
    // HP
    string = string + Integer.toString(pokemon.getHit_points()) + ", ";
    // NAME
    if (pokemon.getName().equals("Farfetch'd")) { string = string + "'Farfetch''d', "; }
    else { string = string + "'" + pokemon.getName() + "', "; }
    // NUMBER
    string = string + Integer.toString(pokemon.getId()) + ", ";
    // SP_ATTACK
    string = string + Integer.toString(pokemon.getSp_attack()) + ", ";
    // SP_DEFENSE
    string = string + Integer.toString(pokemon.getSp_defense()) + ", ";
    // SPEED
    string = string + Integer.toString(pokemon.getSpeed()) + ", ";
    // TYPES
    string = string + "'" + pokemon.getTypes()[0] + "', '" + pokemon.getTypes()[1] + "', ";
    // WEIGHT
    string = string + Double.toString(pokemon.getWeight()) + ", ";
    // GENERATION
    string = string + Integer.toString(pokemon.getGeneration()) + ", ";
    // LEGENDARY
    string = string + Boolean.toString(pokemon.isLegendary());

    return string;
}
```

Challenges Continued :

In addition to the spelling errors we kept getting, some of our characters would come out of the database oddly, mostly accented e-s. We mainly got around this by replacing the crazy characters with the standard version of the letter.

Originally when we were searching for a data collection to use, we were looking for a database that contained the images of the pokemon within it, but that proved much harder to find. If we had found this mythical dataset, we wanted to create a graphical interface where we would show the images of the pokemon returned from the search on buttons instead of our text interface that we implemented.

Part Five : Why we chose this dataset?

We chose this dataset because it was the most diverse: it has data ranging from stats against each type of pokemon to legendary status to pokemon generation. This excited us a lot since it gave us a lot of flexibility in terms of what we could do with the dataset.

Since this was an enormous dataset (41 columns and around 800 rows), we had to do a lot of preprocessing to reduce the number of columns (35) to data which would be most useful. Also, we were very interested in learning about all the information on our favorite pokemons and this dataset had all of the data we would need in a single file.

Part Six: Future Work

Our main goal in the future is to condense the classes. We want to have better code design and make the classes more universal. Right now, our classes are a little scattered and do a lot of things. We want to streamline the process of making a database and making it more general. We would also like to make our code more concise since we could use data structures for various parts of our code.