

Victor Majestic, Jonathan Lutch, James Farmer
CSDS341 - Databases Final Project Report

Table of Contents

I.	Access our Application	2
II.	How to Reproduce the Project	3
III.	Database Description	4
	A. ER Diagram	4
	B. Table Explanations	5
IV.	User Manual	8
V.	Use Cases	10
	A. Jonathan	10
	B. James	10
	C. Victor	11
VI.	Database Structure	13
VII.	Reflection	14

I. Access our Application

This section will outline answers to the following questions:

1) What is the name of your database backup file placed in Canvas? It should have the extension of “.bak”.

The file is “PokemonDatabaseBackup.bak”, without the quotes. It is in the same ZIP file as this report.

2) What is the name of the virtual machine where your project runs and works, csdswinlabxxx where xxx are three digits?

Our VM name is “csdswinlab023”, without the quotes.

3) What is the username for virtual machine - unless you changed it, it will be student.

We did not change the username. It is “student”, without the quotes.

4) What is the password for the username provided in the above step?

The password is “Py84V9XDzVeqKNcT”, without the quotes.

5) What are the instructions for running your code on the virtual machine? That is, how do we test your database application?

See Section IV (User Manual) for further instructions.

6) What is the name of your zip file that contains your Java code? This file should be placed in Canvas.

This report is in the same ZIP file that the Java code is in. The Java file is “SQLExecQuery.java”, without the quotes.

7) When are you meeting with your Teaching Assistant to review and demo your code? Give the date/time and please make certain that your TA is recording your demo.

We are meeting Wednesday, May 3, 2023, at 2:40 PM.

8) Are there any other username/passwords that we need to run your code/database?

No.

II. How to Reproduce the Project

Our database was created using the Microsoft SQL Server Express 2019 Database Management System. The project requires the file “mssql-jdbc-11.2.0.jre11.jar” and JDK 11. We used a Virtual Machine which had all configurations set up for us. We cannot provide instructions on these configurations.

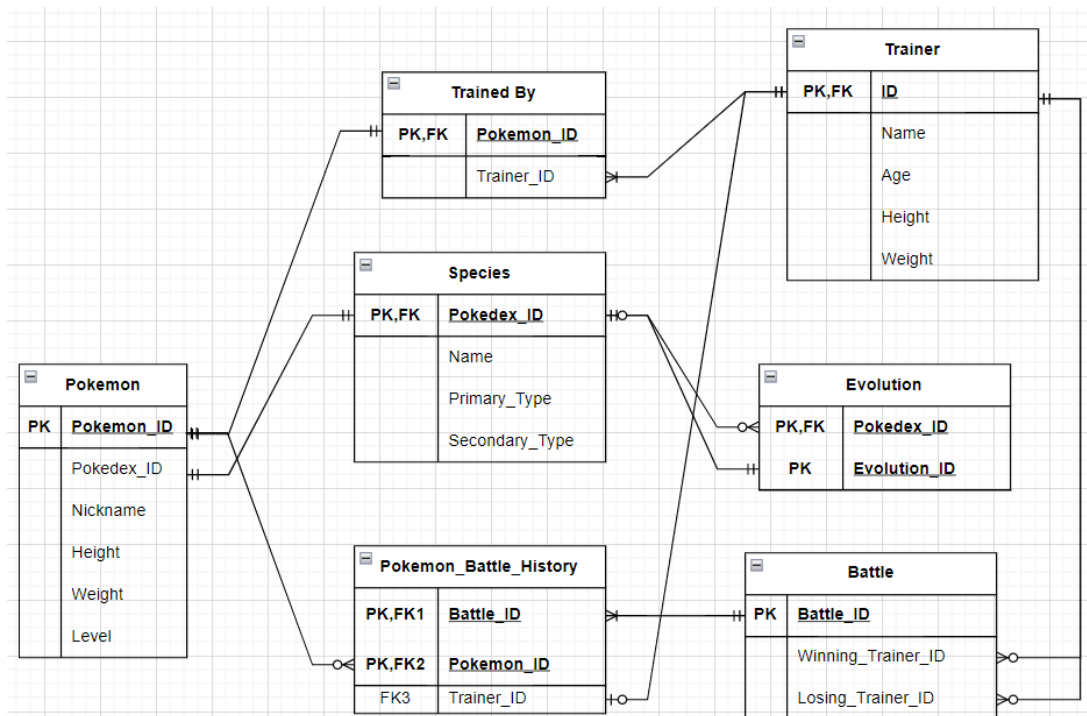
Assuming the configurations are correct (as in, a blank working Virtual Machine without the project), then follow these instructions:

1. Create a new database on Microsoft SQL Server named “pokemon” without the quotes.
2. All SQL statements required to create and populate the database are located in the “sql queries.sql” or “sql queries.txt” file located in the same ZIP file as this report. Open a new query, copy-paste each query in the file, and press “Execute”.
3. Ensure the database itself and each table have permissions given to the user you would like to have access. This is done in Microsoft SQL Server using “Permissions”.
4. The Java file required to run our use cases and User Interface is called “SQLExecQuery.java” without the quotes. It is located in the same ZIP file as this report. Under the method connectToDatabase, modify the user and password to reflect the user. The particular lines of code are on lines 578 and 579.
5. Assuming all other configurations are correct, the project should be good to run. See Section IV (User Manual) for further instructions.

III. Database Description

This database is based on the Pokemon universe. It is able to keep records of Pokemon species, Trainers, individual Pokemon owned by Trainers, and battles. Our ER diagram is below:

IIIa. ER Diagram



Note a couple of minor errors with the diagram:

1. The relationship between Pokemon(Pokedex_ID) and Species(Pokedex_ID). The foreign key is Pokemon(Pokedex_ID) and should be labeled as such. The referenced attribute is Species(Pokedex_ID) and should not be labeled as FK.
2. The relationship between Battle(Winning_Trainer_ID)/Battle(Losing_Trainer_ID) and Trainer(ID). The foreign keys are Battle(Winning_Trainer_ID) and Battle(Losing_Trainer_ID) and should be labeled as such. The referenced attribute is Trainer(ID) and should not be labeled as FK.
3. The relationship between TrainedBy(Trainer_ID) and Trainer(ID). It should be zero-to-many on the side of TrainedBy(Trainer_ID). TrainedBy(Trainer_ID) should also be labeled as FK.
4. The table Pokemon_Battle_History is actually called PokemonBattleHistory in our database.
5. The table Trained By is actually called TrainedBy in our database.
6. The attribute Pokemon(Pokemon_ID) is actually called Pokemon(ID) in our database.

7. The attribute Battle(Battle_ID) is actually called Battle(ID) in our database.

IIIb. Table Explanations

Our database is comprised of seven tables:

1. Species

The Species table will list the general characteristics of a Pokemon species. It will have the following attributes:

- a. Pokedex_ID: This value refers to the number assigned to the Pokemon species according to the National Pokedex. This value is an int, cannot be null, and must be a value between 1 and 1015. For reference, the National Pokedex lists all Pokemon in order (first) by the home region and (second) by the order of its regional Pokedex. The first ID is 1 (Bulbasaur), and the last known ID is 1010 (Iron Leaves), but there are five more unreleased Pokemon that comprise IDs 1011-1015. This value is the primary key.
- b. Name: The name of the Pokemon species. This value is varchar and cannot be null. For example, Bulbasaur is a Pokemon species.
- c. Primary_Type: The first elemental type of a Pokemon species. This value is varchar and cannot be null. It also must be a valid Pokemon type, which includes Normal, Fire, Water, Grass, Bug, Ice, Ground, Rock, Electric, Fighting, Poison, Flying, Psychic, Ghost, Dark, Dragon, Steel, and Fairy.
- d. Secondary_Type: The second elemental type of a Pokemon species. This value is varchar and can be null. It must also be a valid type and unique from the Primary_Type.

2. Evolution

The Evolution table keeps track of the next evolution of the Pokemon. For instance, the Species Charmander evolves into Charmeleon, and Charmeleon evolves into Charizard. It will have the following attributes:

- a. Pokedex_ID: Refers to the Pokedex ID of the Species before it evolves. This value is an int and cannot be null. This attribute is one half of the primary key and is a foreign key referencing the Pokedex_ID attribute of the Species table.
- b. Evolution_ID: This value refers to the Pokedex ID of the Species after it evolves. This value is an int and cannot be null. This attribute is one half of the primary key and is a foreign key referencing the Pokedex_ID attribute of the Species table.

3. Trainer

The Trainer table keeps track of Trainers, or humans taking care of and training the Pokemon. It will have the following attributes:

- a. ID: A unique value that identifies a Trainer. This value is an int and cannot be null. This is the primary key.
- b. Name: The name of the Trainer. This value is varchar and cannot be null.
- c. Age: The age of the Trainer. This value is a tinyint and cannot be null.
- d. Height: The height of the Trainer (in inches). This value is an int and cannot be null.
- e. Weight: The weight of the Trainer (in pounds). This value is an int and cannot be null.

4. Pokemon

The Pokemon table refers to an individual, unique Pokemon. It will have the following attributes:

- a. ID: A unique value that identifies a Pokemon. This value is an int and cannot be null. This is the primary key.
- b. Pokedex_ID: Refers to the Pokedex ID of the Species that the Pokemon is. This value is an int and cannot be null. This attribute is a foreign key referencing the Pokedex_ID attribute in the Species table.
- c. Nickname: The name given to the Pokemon. This value is varchar and cannot be null.
- d. Height: The height of the Pokemon (in inches). This value is an int and cannot be null.
- e. Weight: The weight of the Pokemon (in pounds). This value is an int and cannot be null.
- f. Level: A value between 1 and 100 that indicates how experienced the Pokemon is. If the Level is closer to 1, the Pokemon is less experienced, and if the Level is closer to 100, the Pokemon is more experienced. This value is a tinyint, cannot be null, and must be between 1 and 100.

5. TrainedBy

The TrainedBy table refers to the relationship between a Pokemon and a Trainer. It will have the following attributes:

- a. Pokemon_ID: Refers to the unique ID of the Pokemon. This value is an int and cannot be null. This is the primary key, and a foreign key referencing the ID attribute of the Pokemon table.
- b. Trainer_ID: Refers to the unique ID of the Trainer. This value is an int and can be null. This is a foreign key referencing the ID attribute of the Trainer table.

6. Battle

The Battle table keeps a record of the battles that occur between two Trainers. In a Battle, a Trainer uses at least one and up to six Pokemon against another Trainer's Pokemon. It will have the following attributes:

- a. ID: A unique value that identifies a Battle. This value is an int and cannot be null. This is the primary key.
- b. Winning_Trainer_ID: The ID of the Trainer who won the Battle. This value is an int and cannot be null. This attribute is also a foreign key referencing the ID attribute in the Trainer table.
- c. Losing_Trainer_ID: The ID of the Trainer who lost the Battle. This value is an int and cannot be null. This attribute is also a foreign key referencing the ID attribute in the Trainer table.

7. PokemonBattleHistory

The PokemonBattleHistory table tracks the Pokemon that participate in a Battle. It will have the following attributes:

- a. Battle_ID: Refers to the Battle that the Pokemon participated in. This value is an int and cannot be null. This is one half of the primary key and is a foreign key referencing the ID attribute in the Battle table.
- b. Pokemon_ID: Refers to the Pokemon that participated in the Battle. This value is an int and cannot be null. This is one half of the primary key and is a foreign key referencing the ID attribute in the Pokemon table.
- c. Trainer_ID: Refers to the Trainer that owns the Pokemon who participated in the battle. This value is an int and cannot be null. This is a foreign key referencing the ID attribute in the Trainer table.

IV. User Manual

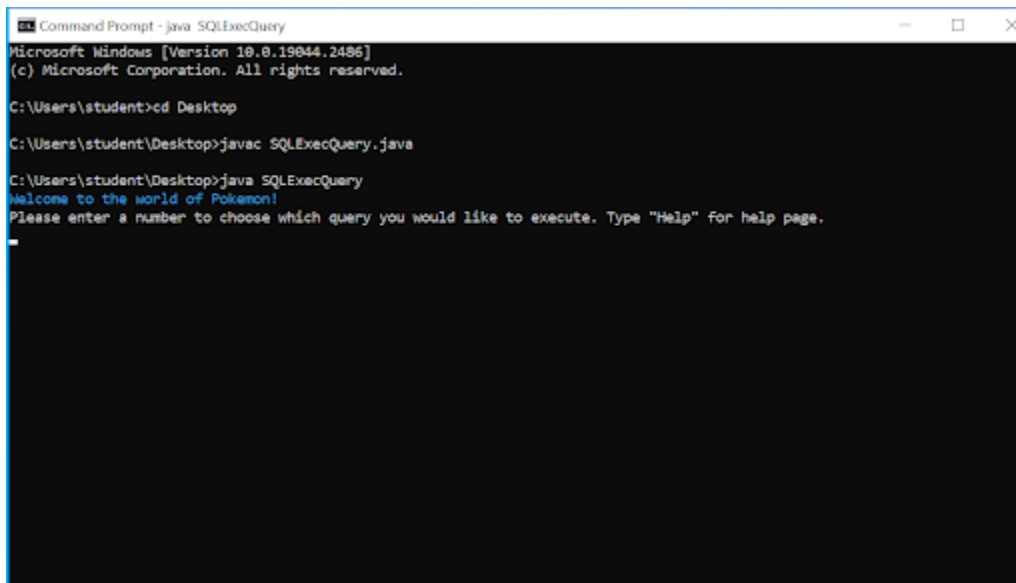
For this section, note that all inputs will be included in quotation marks, but do not type any quotation marks as input.

The User Interface (UI) can be loaded in one of two ways:

In the Command Prompt terminal, ensure you are in the same directory where the `SQLExecQuery.java` file is stored. Then, type the following two commands: “`javac SQLExecQuery.java`” and “`java SQLExecQuery`” in order.

Alternatively, open the `SQLExecQuery.java` file in Visual Studio Code. In the Terminal, ensure you are in the same directory where the `SQLExecQuery.java` file is stored. Then, type the following two commands: “`javac SQLExecQuery.java`” and “`java SQLExecQuery`” in order.

You will know the UI is loaded when you see the following. The first image is from the Command Prompt and the second is from VSCode.

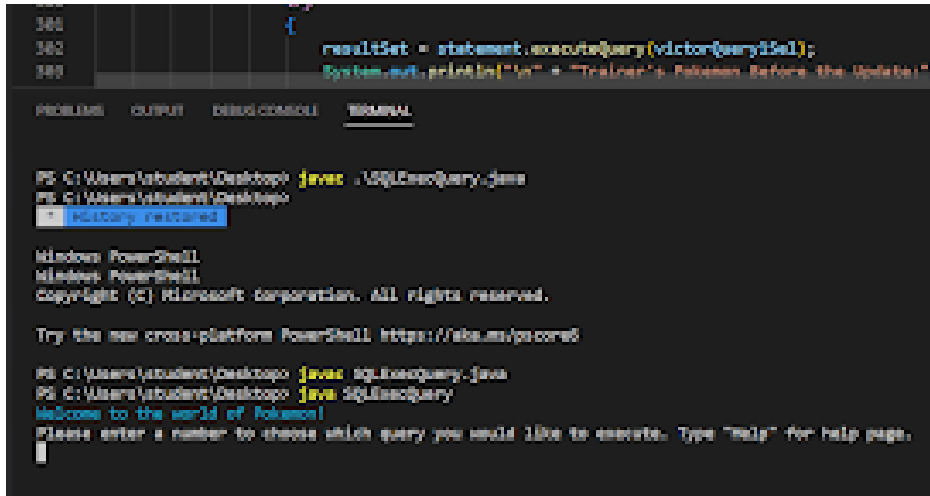


```
Command Prompt - java SQLExecQuery
Microsoft Windows [Version 10.0.19044.2486]
(c) Microsoft Corporation. All rights reserved.

C:\Users\student>cd Desktop

C:\Users\student\Desktop>javac SQLExecQuery.java

C:\Users\student\Desktop>java SQLExecQuery
Welcome to the world of Pokemon!
Please enter a number to choose which query you would like to execute. Type "Help" for help page.
_
```

The screenshot shows a Java IDE with a code editor at the top and a terminal window at the bottom. The code editor displays a Java snippet with line numbers 340, 341, 342, and 343. Line 342 contains the statement `resultSet = statement.executeQuery(victorQuery);` and line 343 contains `System.out.println("\n" + "Trainer's Pokemon Before the Update:");`. The terminal window shows the command prompt with the following text:

```

PS C:\Users\student\Desktop> java .\SQLExecQuery.java
PS C:\Users\student\Desktop> 1
Welcome to the world of Pokemon!
Please enter a number to choose which query you would like to execute. Type "Help" for help page.

```

The best place to start is by typing “Help” (not case-sensitive). It will bring up the list of valid user inputs. Typing “1” or “2” will run Jonathan’s use cases, “3” or “4” will run James’s use cases, and “5” or “6” will run Victor’s use cases.

Additionally, typing “Species,” “Pokemon,” “Trainer,” “TrainedBy,” “Evolution,” “Battle,” or “PokemonBattleHistory”/“PBH” will run a “SELECT * from x” query where x is the respective table. None of these inputs are case-sensitive.

Entering anything else will give an error message and suggest a different input. Type “Quit” (not case-sensitive) to exit the program.

See Section V (Use Cases) for a description of what each use case does.

V. Use Cases

Va. Jonathan

Jonathan's first use case involves changing the Nickname of all Pokemon belonging to a Trainer, and can be implemented by typing "1". Currently, this is only Misty's Pokemon (Trainer Misty has an ID of 2), and all her Pokemon's Nicknames will be changed to "JonathansPokemon". No user input is required. This use case makes use of the SELECT and UPDATE queries. First, it will select all of Misty's Pokemon and display them in a table. Then, the Nicknames will be updated to match "JonathansPokemon". Finally, the first select statement is executed again so the user can see the updated table. This use case uses a transaction when updating.

The queries used are:

- `SELECT Nickname FROM Pokemon JOIN TrainedBy ON Pokemon.ID = TrainedBy.Pokemon_ID JOIN Trainer ON Trainer.ID = TrainedBy.Trainer_ID WHERE Trainer.Name = 'Misty'`
- `UPDATE Pokemon set Nickname = 'JonathansPokemon' FROM Pokemon JOIN TrainedBy ON Pokemon.ID = TrainedBy.Pokemon_ID JOIN Trainer ON Trainer.ID = TrainedBy.Trainer_ID WHERE Trainer.Name = 'Misty';`

Jonathan's second use case involves selecting all Pokemon (not Species) above a certain Level who have an Evolution. This can be implemented by typing either "2" or "2sp". The former input assumes you are looking for Pokemon with the Water Primary_Type, any Secondary_Type, and Level 40. The latter input requires three user inputs: a Level, Primary_Type, and Secondary_Type. A Primary_Type or Secondary_Type input value of "*" indicates any type. An invalid input will return an error message. This use case only uses the SELECT query. The "2sp" input makes use of an internal SQL stored procedure.

The default query is:

- `SELECT * FROM Pokemon JOIN Species ON Pokemon.Pokedex_ID = Species.Pokedex_ID JOIN Evolution ON Pokemon.Pokedex_ID = Evolution.Pokedex_ID WHERE Level > 40 AND Primary_Type = 'Water'`

Vb. James

James's first use case involved adding a record of a battle between two Trainers. Here, we assume each Trainer is only using one Pokemon. This can be implemented by typing "3". This requires four user inputs: the winning Trainer's ID, the winning Pokemon's ID, the losing Trainer's ID, and the losing Pokemon's ID. An invalid input will return an error message. This use case only makes use of the INSERT query. Three total queries are run. First, the Battle table

is updated with the IDs of the winning and losing Trainers. Then, the PokemonBattleHistory table is updated with the IDs of the winning Pokemon and Trainer. Finally, the PokemonBattleHistory table is updated with the IDs of the losing Pokemon and trainer. This use case involves transactions.

The queries used are below. Here, assume the winning Trainer ID is 1, the losing Trainer ID is 2, the winning Pokemon ID is 3, and the losing Pokemon ID is 4. However, the queries will match the user inputs. Also assume the Battle_ID being inserted is 5, but the database will automatically use the next available Battle_ID. We do not have a check to ensure the Pokemon is owned by the Trainer, because Pokemon can be traded (a Pokemon can be TrainedBy more than one Trainer, but not at the same time).

- INSERT INTO Battle (Winning_Trainer_ID, Losing_Trainer_ID) VALUES (1, 2)
- INSERT INTO PokemonBattleHistory (Battle_ID, Pokemon_ID, Trainer_ID) VALUES (5, 1, 3)
- INSERT INTO PokemonBattleHistory (Battle_ID, Pokemon_ID, Trainer_ID) VALUES (5, 2, 4)

James's second use case involves deleting a Trainer from the database. This can be implemented by typing "4". This use case requires one input, the Trainer_ID to be deleted. An invalid input will return an error message. This use case uses the SELECT and DELETE queries. Three total queries are run. First, all Pokemon TrainedBy the deleted Trainer is selected, and the user is informed that all the Pokemon are released to the wild. Then, the Battle table is updated so that any rows containing instances of the deleted Trainer are removed. Finally, the Trainer itself is deleted. At this point, the Trainer_ID of the deleted Trainer in the TrainedBy table is set to null. This use case involves transactions.

The queries used are below. Here, assume the deleted Trainer has ID 1. However, the queries will match the user input.

- SELECT Pokemon.Nickname FROM Pokemon JOIN TrainedBy on Pokemon.ID = TrainedBy.Pokemon_ID WHERE TrainedBy.Trainer_ID = 1
- DELETE FROM Battle WHERE Winning_Trainer_ID = 1 OR Losing_Trainer_ID = 1
- DELETE FROM Trainer WHERE ID = 1

Vc. Victor

Victor's first use case involves leveling up Pokemon. Assume that if a Trainer wins a battle, all of their Pokemon who participated level up once. This can be implemented by typing "5". This requires one user input, a Battle_ID. An invalid input will return an error message. This use case makes use of the SELECT and UPDATE queries. First, the database will select all Pokemon who won the Battle with the respective Battle_ID. Then, the levels of those Pokemon are increased by

1. Finally, the first select statement is executed again so the user can see the updated table. This use case uses a transaction when updating.

The queries used are as follows. Note that this assumes the Battle_ID is 1, but the value will match the user input:

- `SELECT Pokemon.ID, Nickname, Level from Pokemon JOIN PokemonBattleHistory ON Pokemon.ID = PokemonBattleHistory.Pokemon_ID JOIN Battle ON PokemonBattleHistory.Battle_ID = Battle.ID where PokemonBattleHistory.Trainer_ID = Battle.Winning_Trainer_ID and Battle_ID = 1`
- `UPDATE Pokemon set Level = Level + 1 from Pokemon JOIN PokemonBattleHistory ON Pokemon.ID = PokemonBattleHistory.Pokemon_ID JOIN Battle ON PokemonBattleHistory.Battle_ID = Battle.ID where PokemonBattleHistory.Trainer_ID = Battle.Winning_Trainer_ID and Battle_ID = 1`

Victor's second use case involves selecting all Trainers who have won a user-specified value number of Battles or more. This can be implemented by typing "6". This requires one user input, a number referring to the minimum number of wins. An invalid input will return an error message. This use case only uses the SELECT query. Then, the database returns a list of all Trainers who meet the criteria.

The query used is below. Note that this assumes the minimum number of wins is 1, but the value will match the user input:

- `SELECT * from Trainer WHERE (SELECT COUNT(*) from Battle where Trainer.ID = Battle.Winning_Trainer_ID) >= 1`

VI. Database Structure

In terms of functional dependencies/normalization, the database represents BCNF for the most part. This means all functional dependencies are completely reliant on the primary key of each table. The exception is in the Species table, in which $\text{Name} \rightarrow \text{Primary_Type}$ and Secondary_Type .

As the database uses the SQL Server DBMS, B-Trees are used for indexing. Triggers are not used.

VII. Reflection

While we only have six use cases, our database is set up to where any of the following use cases can be implemented:

1. Insert a Trainer into the database.
2. Insert a Pokemon into the database.
3. Void a Battle and remove it from the database. This would also require updating the PokemonBattleHistory table.
4. Remove a Pokemon from the database.
5. Select all Species of a particular type, whether Primary, Secondary, or both.
6. Select all Pokemon of a particular Level, above a Level, or below a Level.
7. Select all Battles that a Trainer participated in.
8. Select all Battles that a Trainer won or lost.
9. Select all Species that can evolve or Species that cannot evolve.
10. Give a user-specified number of Rare Candies to a Pokemon—this increases the Pokemon’s Level by that number.
11. Select all Battles that a Pokemon participated in.
12. Remove the Nickname of a Pokemon.
13. Select all Trainers above, below, or equal to a particular Age, Height, or Weight.
14. Select all Pokemon above, below, or equal to a particular Height or Weight.
15. Select all Species from a particular region (for instance, Pokedex_ID between 1 and 151 are from the Kanto region, between 152 and 251 are from the Johto region, etc.).
16. Select all Pokemon that were used by a Trainer other than their current Trainer in a Battle.

More use cases could be added to this list.

Possible additional tables that could be added are those which keep track of Pokemon Moves. Each Move has a Name, a Base Power, a Type, and an Accuracy. The weakest Base Power move is Constrict (10), and the strongest is Explosion (250). The lowest Accuracy Move is 30, which are moves that one-hit-knockout. The highest Accuracy Move is infinity, as in, the Move cannot miss no matter the circumstances. The next highest accuracy is 100. Another table could track the compatibility of Species with Moves. Here, we assume that all Moves can do damage, but some Moves only inflict status effects.

In terms of individual contribution, each member was responsible for their two use cases. In addition, Jonathan created the initial UI in Java and tables in SQL. Victor took the lead with the reports, implemented the “SELECT *” statements, and populated the database. James hosted the Virtual Machine we used to create this project and was responsible for the more complicated use

cases (involving insert/delete). He also modified the permissions required for our Java interface to function properly.

To the TAs or whoever is grading this project: If there is an issue with the submission, including the absence of or errors in any deliverables, please do not hesitate to contact us:

- Victor Majestic - vjm38@case.edu
- Jonathan Lutch - jal357@case.edu
- James Farmer - jlf124@case.edu