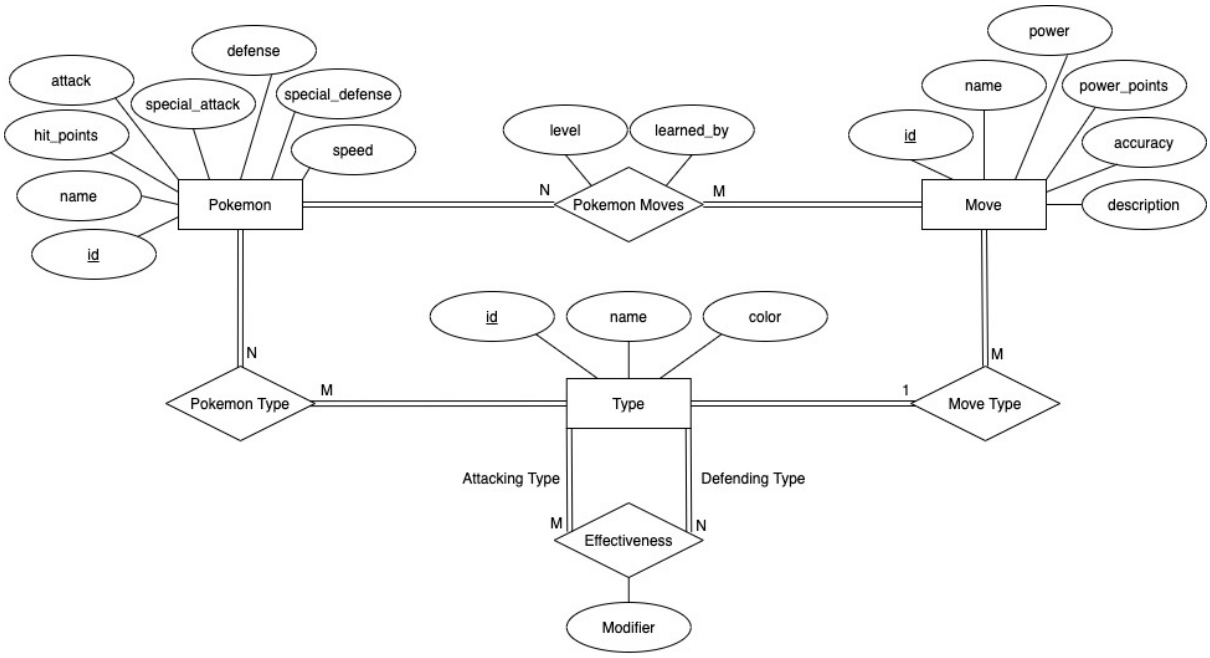
Programming Assignment 2 Report

Student: | Olof Enström - oe222fh@student.lnu.se

1. Project Idea

For this assignment I have designed and implemented a website to store information about Pokémon. I downloaded the data from https://www.kaggle.com/ and manually created the data in “type.csv” and “effectiveness.csv”. The website enables users to search for Pokémons and moves and view the information about them. The information about the Pokémon are their name, base stats, type, type weakness/strength and what moves they can learn. The move information are the name, power, accuracy, power points, type, type weakness/strength, what Pokémons that can learn the move and a description of the move. This is a nice tool to have for people playing Pokémon games as it gives the user any easy way to get information about the Pokémon they catch.

2. Schema Design



I have three entity sets, Pokemon, Move and Type. The Pokemon set has the attributes id, as primary key, name, hit_points, attack, defense, special_attack, special_defense and speed. As a Pokémon can have one or two types and multiple Pokémons can have the same type, I have a many-to-many relation, Pokemon Type, between the Pokemon entity set and the Type entity set. A Pokémon can also have multiple moves and multiple Pokémons can have the same moves, so I have a many-to-many relation, Pokemon Moves, between

the Pokemon and Move entity sets. This relation also have two attributes, level and learned_by. These two determines how and what level a Pokémon learns a move.

The entity set Type only has three attributes, id, name and color. Where id is the primary key. The type set has a relation, Effectiveness, with it self. On this relation I added the modifier attribute which decides the effectiveness of the Attacking Type against the Defending Type. The Type entity set also has a relation with the Move entity set. As a Move only can have one Type, it's a one-to-many relationship from Type to Move.

Finally we have the Move entity set with the attributes id, name, power, accuracy, power_points and description. All the relations have already been described for this one.

3. SQL Queries

I have a couple more queries but some are boring and some are a bit like the ones described below. I decided to make three views to make it easier for the other queries. So let's start with them.

Q: **Create a view of all Pokémons with their types and the types colors.**
The following query is a multirelation query and uses two *JOINs.* The query should create a view of all the Pokémons, with their data, the type of the Pokémon and the color of the type. I join table *pokemon* on table *pokemon_type* by matching the *pokemon.id* to the foreign key *pokemon_type.pokemon_id* and on the table *type* by matching the *type.id* to the foreign key *pokemon_type.type_id.*

```
CREATE OR REPLACE VIEW pokemon_with_type AS
  SELECT
    pokemon.*,
    type.name AS type,
    type.color
  FROM pokemon
  JOIN pokemon_type ON pokemon_id = pokemon.id
  JOIN type ON type_id = type.id;
```

Q: **Create a view of all Effectiveness modifiers with the name of the types and color.**
The following query is a multirelation query and uses two *JOINs.* The query should create a view of all Types as attacking and defending, with colors, and with the corresponding modifier. I join table *effectiveness* on table *type as t1* by matching the *t1.id* to the foreign key *attacking_type_id* and on the table *type as t2* by matching the *t2.id* to the foreign key *defending_type_id.*

```
CREATE OR REPLACE VIEW effectiveness_with_type AS
  SELECT
    effectiveness.attacking_type_id AS attacking_id,
```

```
    t1.name AS attacking_name,
    t1.color AS attacking_color,
    effectiveness.modifier,
    effectiveness.defending_type_id AS defending_id,
    t2.name AS defending_name,
    t2.color AS defending_color
  FROM effectiveness
  JOIN type t1 ON t1.id = attacking_type_id
  JOIN type t2 ON t2.id = defending_type_id;
```

Q: **Create a view of all Moves with their type and type color.**
The following query is a multirelation query and uses two *JOINs.* The query should create a view of all Moves with their data, type and type color. I join table *move* on table *move_type* by matching the *move.id* to the foreign key *`move_id`* and on the table *type* by matching the *type.id* to the foreign key *`type_id`.*

```
CREATE OR REPLACE VIEW move_with_type AS
  SELECT
    move.id, move.name, power, accuracy, power_points,
    description, type.name AS type, type.color
  FROM move
  JOIN move_type ON move.id = move_id
  JOIN type ON type.id = type_id;
```

Q: **List Pokémon with data, sum of base stats and type,color as JSON_OBJECT.**
The following query is a single relation query but uses the view *pokemon_with_type.* I select all the data about the Pokémons, I add all the base stat attributes together into a *sum* attribute. As a Pokémon can have two types that Pokémon appear twice in the view *pokemon_with_type.* To only get one result of the Pokémon but still get both *types* and *colors* I use the aggregate function *JSON_OBJECTAGG* and I group by *id.* This gives me one result for each Pokémon with *type* and *color* as a JSON key value pair string in a single attribute. I can also pass in a *WHERE* clause with a Pokémon name or ID (marked with {cond} in the query) which will display only one Pokémon instead of all.

```
SELECT
  id, name, hit_points, attack, defense, special_attack,
  special_defense, speed, (hit_points + attack + defense +
  special_attack + special_defense + speed) AS sum,
```

```
  JSON_OBJECTAGG(type, color) AS type_color
FROM pokemon_with_type
{cond}
GROUP BY id;
```

Q: **List all moves that a certain Pokémon can learn.**
The following query is a multirelation query and uses a *JOIN* and the view *move_with_type.* The query should list all Moves with their data, type and type color that a Pokémon can learn. I join table/view *move_with_type* on table *pokemon_move* by matching the *id* to the foreign key *move_id.* I pass in a parameter with the id of the Pokémon (marked with {?} in the query) which will only select moves for that Pokémon. I also order the result by the level the Pokémon learns the move and how they learn it.

```
SELECT * FROM move_with_type
JOIN pokemon_move ON id = move_id
WHERE pokemon_id = {?}
ORDER BY level, learned_by DESC;
```

Q: **List Effectiveness modifiers for the Pokémon getting attacked by other types.**
The following query is a single relation query but uses the view *effectiveness_with_type.* The query should list the modifiers for types attacking the Pokémons combination of types. with their data, type and type color that a Pokémon can learn. I pass in a parameter with the *name* of the defending type (marked with {?} in the query) I can also pass in an other *defending_name = {?}* (marked with {where} in the query) if the Pokémon got two types. I group by the *attacking_id.* The attacking color will be the same for both types of the Pokémon, so I just use MAX() to get the color so I can use GROUP BY. If the Pokémon got two types the aggregate function *JSON_ARRAYAGG* gives me a JSON array containing the attacking types modifier for the Pokémons two types. I can then multiply the two values in Python before displaying them.

```
SELECT
  attacking_id, attacking_name,
  MAX(attacking_color) AS attacking_color,
  JSON_ARRAYAGG(modifier) AS modifier
FROM effectiveness_with_type
WHERE defending_name = {?} {where}
GROUP BY attacking_id;
```

Q: **List all Pokémons that can learn a certain move.**
The following query is a multirelation query and uses a *JOIN.* The query should list all Pokémons that can learn a certain move. I join table *pokemon_move* on table *pokemon* by matching the *id* to the foreign key *pokemon_id.* I pass in a parameter with the id of the move (marked with {?} in the query) which will only select Pokémons that can learn that move.

```
SELECT * FROM pokemon_move
JOIN pokemon ON pokemon_id = id
WHERE move_id = {?};
```

4. Discussion and Resources

The project uses the built json library and csv library for reading the data. It uses the external libraries Flask and mysql.connector. Please check the README.md in GitHub for installation details.

The data I downloaded from Kaggle were not split the way I wanted. So I made a script to fix and split the data in to csv files so it could be easily inserted in to the database. It also didn't have data for the effectiveness between types, so I manually typed that in to the file "effectiveness.csv".

The data is split up very much which gives me good flexibility and is future proof. The negative thing is that some queries becomes pretty long and tedious to write. That's why I decided to make three views. It makes my other queries shorter and simpler and in this case it's worth it even though it all becomes a bit slower as more queries have to be run.

There is a ton more data and features that can be added to make this tool more complete. Like more regions and their Pokémons (I have only included the Kanto region), evolvements of Pokémons, eggs, gyms, Pokémon League, more data and information about the Pokémons. But I felt like that's out of the scope of this project.

Source code: https://github.com/oenstrom/pa2-1dv503
Video demonstration: https://youtu.be/6Up__U0ARNo