SQL and Flask Pokemon Project

Intro

Now, that we learned the basic SQL features, and integrated it with python it is time to add the server layer.
In this mini project we will upgrade the pokemon exercise and create a minimal pokemon API.

While building the project we will practice the following topics:

- Simple server
- HTTP routes using GET, POST, PUT, DELETE methods
- Error handling and HTTP status codes
- Working with an external API: pokeAPI

Prerequisite - Pokemon data migration exercise

This exercise is built on a previous sql in python ex, that migrates pokemon data into a DB and define queries over the DB.
Please finish first the exercise “1-SQL in Python_pymysql”
If you haven't done the exercise, Here are the instructions, for your convenience. You must complete it before moving on.

Previos exercise Brief

PokeCorp is a company that tracks pokemon and their trainers around the world.

Until this day, they've been storing all their data together, in a single JSON file that looks like this:

```
[{
    "id": <pokemon_id>,
    "name": <pokemon_name>,
    "type": <pokemon_type>,
    "height": <pokemon_height>,
    "weight": <pokemon_weight>,
    "ownedBy": [
      {name: <trainer_name>, town: <trainer_town>},
      ...
    ]
},
...]
```

The file has 151 pokemon in it. Each pokemon has some data, as well as an `ownedBy` field.

The `ownedBy` field is an array of objects, where each object represents a trainer that owns this pokemon - note that this array might be empty.

You don't know exactly how many there are, but you know there are fewer than 50 trainers. Of course, each trainer can own more than one pokemon.

Likewise, you don't know how many pokemon types there are, but you know it is fewer than 151.

PokeCorp has hired you and a Senior Database Administrator (DBA) to migrate their data to an SQL database.

The DBA is very busy, and she left you with pretty much the entire task.

Start by planning how the DB will look like. Think about which tables you need, what are the relationships between the tables, and what are the properties for each table.

You should create the tables using plain SQL (in some `.sql` file), and then do all the `INSERT`s in python using the `pymysql` package.

Once you're done, write functions for the following queries.

For the following queries, the query itself should be in SQL, but the wrapping function in python. Obviously, use `pymysql`

Query Exercise 1

Write a function that receives a pokemon type, and returns all of the pokemon names with that type.

For instance, `findByType("grass")` should return `["bulbasaur", "ivysaur", "venusaur", "oddish", ...]`

Query Exercise 2

Write a function that receives the name of a pokemon, and returns the names of all the trainers that own it, or an empty array if no one owns it.

For instance, `findOwners("gengar")` should return `["Misty", "Wallace", "Gary", "Plumeria"]`

Query Exercise 3

Write a function that receives the name of a trainer, and returns the names of all the pokemon he or she owns.

For instance, `findRoster("Loga")` should return `["metapod", "raticate", "spearow", "pikachu", "machoke", "machamp", "weepinbell", "cloyster", "kabuto"]`

Instructions

We want to build a pokemon API.
Start by creating the skeleton for the server and make sure it works.

A small simplification, in our project each trainer can only have 1 pokemon of each kind. So if a trainer Bibi has a spearow pokemon, she cannot have anymore spearow pokemons.

Before you start implementing the code, take a minute (or even 30) to plan your DB structure and your API endpoints. Read all the instructions, understand the data, and prepare a design.

Important note: in the data that you got, each pokemon had only 1 type, but in fact pokemons can have more than 1 type.
Please make the necessary updates, so that your code support the ability of a pokemon to have multiple types.
So, in order to fix this problem we will add a route that will update a pokemon's types. The route will get the pokemon name as a parameter and will access the external pokeAPI to get the types and then update the DB.

So, go ahead and the route:

Update pokemon types: makes sure all the pokemon types are stored according to the information in the PokeAPI.

We will start by embedding some of the queries we wrote in the previous exercise and make them available through the API.

1. Get pokemons by trainer: get all the pokemons of a given owner
2. Get trainers of a pokemon: get all the trainers of a given pokemon

Now we will wan to add some basic abilities:

1. Adding a new pokemon: adds a new pokemon with the following information: id, name, height, weight, types (all of them).
2. Get pokemon by type - returns all pokemons with the specific type. Note that even though we implemented this query you might need to edit it due to the type change.
3. Delete pokemon

Enough warming up, let's dive in.
As you know (or will learn now) pokemons have the ability to evolve into an upgraded version. We will add a route that will make a specific pokemon of a specific trainer evolve.
For that you will need to work with the PokeAPI. In order to know to how a certain pokemon will evolve follow these steps:

1. Get the info of a specific pokemon.
2. From the pokemon general info, get the species url.
3. Get the info of the species, by making a request to the species url
4. From the species info get the evolution chain url
5. Get the info of the evolution chain, by making a request to the evolution chain url
6. From the evolution chain info get the chain item

7. Scan the chain item to find what is the next form of your pokemon. (make sure to cover all cases)
8. You should end up with the name of the evolved pokemon.
9. Update the DB accordingly. (think what needs to be updated)

For example, charmander evolves to charmeleon.

To conclude, these should be your routes:

1. Update pokemon types
2. Add pokemon
3. Get pokemons by type
4. Get pokemons by trainer
5. Get trainers of a pokemon
6. Evolve (pokemon x of trainer y)
7. delete pokemon of trainer

Good, so now you have all the technicality and it is time to focus on the important things, so pay attention.

1. Error handling: think about what can go wrong and how to handle it.
    a. Make sure your app doesn't crash
    b. Return an informative error message
2. Use the correct HTTP status code when you return a response
3. Clean code - make sure your code is clean, consistent and readable
4. Design - think about the code structure, do you need to break your code down? extract functions, split to different files etc...
5. Naming conventions

Tests

Get pokemons by type

Add to the DB (using SQL only) the type of eevee (id 133): normal.

Now validate:

get pokemons by type: normal => validate eevee is there

Send another request to update eevee's types

Result: You should make sure the same types were not re-added, and that your server did not crash.

Add pokemon

Add the pokemon yanma.

You can use the api to get all the needed pokemon details.

Note that yanma has 2 types: bug, flying. Let's check if we can get yanma by it's 2 types:

get pokemons by type: bug => validate yanma is there

get pokemons by type: flying => validate yanma is there

Now, try to add yanma again.

Make sure your server doesn't crashes and returns the correct error message.

Update pokemon types

Let's update the types of pokemon venusaur. If you check the PokeAPI you will see he has 2 types: grass and poison.
Now let's verify:

- get pokemons by type: poison => validate venusaur is there
- get pokemons by type: grass => validate venusaur is there

Get pokemons by owner

Get all of Drasna's pokemons.
You should get:
["wartortle", "caterpie", "beedrill", "arbok", "clefairy", "wigglytuff", "persian", "growlithe", "machamp", "golem", "dodrio", "hypno", "cubone", "eevee", "kabutops"]

Get owners of a pokemon

Get all owners of charmander.
You should get:
["Giovanni", "Jasmine", "Whitney"]

Evolve

- Make a request to evolve Whitney pokemon named pinsir
    - Result: pinsir pokemon can not evolve, so you should return the user a relevant Error message.
- Make a request to evolve Archie pokemon named spearow
    - Result: Archie does not have a spesarow pokemon, so you should return the user a relevant Error message.
- Make a request to evolve Whitney pokemon named oddish
    - Result: oddish should evolve to gloom, and you should return the user a relevant message.
    - Make the same request again: evolve Whitney pokemon named oddish
        - Result: since oddish evolved whitney should not have an oddish pokemon anymore. User should receive a relevant Error message.
    - Get all of Whitney's pokemons.
        - Result: you should see gloom in the list
- For the next test make sure that owner Whitney has pokemons: pikachu and raichu. Now send a request to evolve pikachu
    - Result: since pikachu evolves into raichu and Whitney already has that pokemon, your code should notice it and do nothing