# SQL and FastAPI 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 project we will dive into the world of Pokemons.

Here are some Pokemon species:



The pokemon project is a backend project, combining a python server, an SQL DB and using an external API. The project is based on the Japanese Pokemon video game. The API allows getting information about the different pokemons and managing the state of the pokemon trainers.

In the project you are required to understand each component and the communication between the different components, on top of investigating an external API, and building your own 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](https://pokeapi.co/)

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.

# Part 1 - Pokemon data migration

This exercise is built on a sql in python ex, that migrates pokemon data into a DB and define queries over the DB.

Here are the instructions.

## 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 trainers 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 species there are, but you know it is fewer than 151.

PokeCorp has hired you to migrate their data to an SQL database.

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 INSERTs in python using the pymysql package.

## Important note

In the data that you got, there was a mistake with the pokemon types. You will need to fix this problem. Use the external API to get the correct types for each pokemon.

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.

## 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, findPokemonsOfTrainer("Loga") should return ["metapod", "raticate", "spearow", "pikachu", "machoke", "machamp", "weepinbell", "cloyster", "kabuto"]

# Part 2 - 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.

Your API should support the following operations:

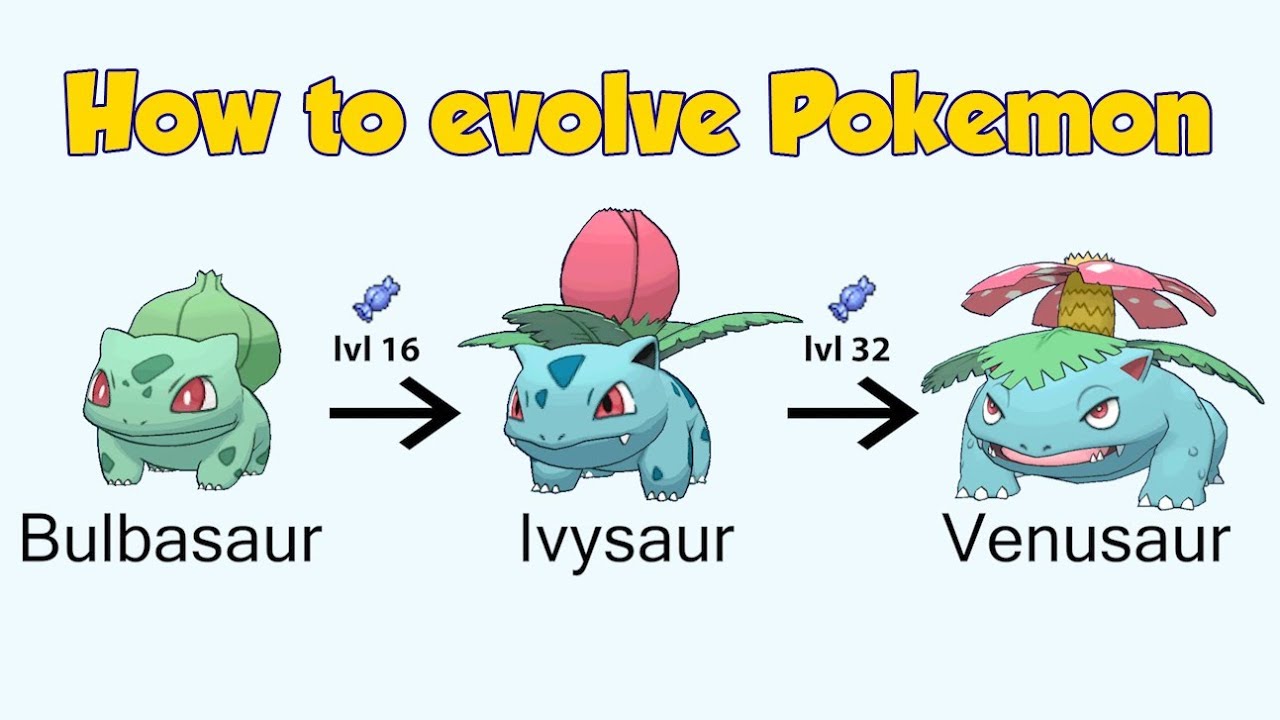
1. Add new pokemon species: adds a new pokemon species with the following information: id, name, height, weight, types (all of them). You can get the information from the external API or add it yourselves.
2. Get pokemons by type: returns all pokemons with the specific type
3. Get pokemons by trainer: get all the pokemons of a given owner
4. Get trainers of a pokemon: get all the trainers of a given pokemon
5. delete pokemon of trainer
6. add pokemon to a trainer: when a trainer catches a pokemon and train it the pokemon become his.
7. Evolve (pokemon x of trainer y)

## Evolution

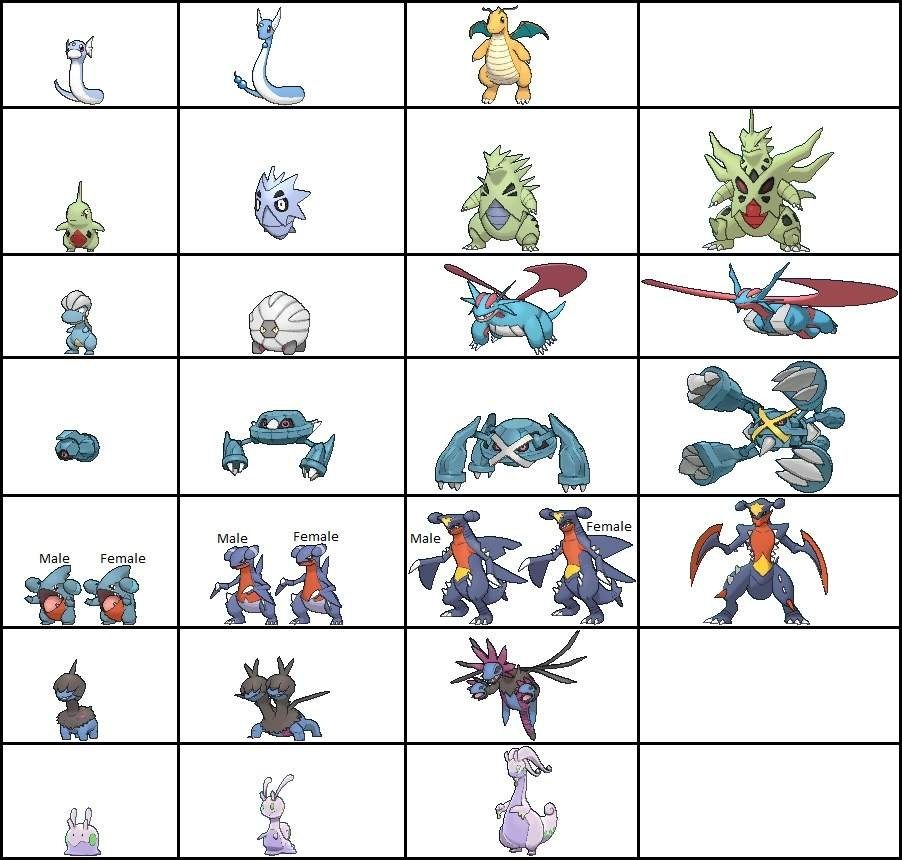
Enough warming up, let's dive in.

As you know (or will learn now) pokemons have the ability to evolve into an upgraded version.

Here are some examples of pokemon evolution:



Some more evolution examples:



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 in order to find what the next form of your pokemon is. (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. Add pokemon
2. Get pokemons by type
3. Get pokemons by trainer
4. Get trainers of a pokemon
5. Evolve (pokemon x of trainer y)
6. delete pokemon of trainer

## Error Handling and Clean Code

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.
   1. Make sure your app doesn't crash
   2. 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

Use postman to test your API.

## Get pokemons by type

Now validate:

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

## Add pokemon

Add the pokemon yanma.

You can use the [api](https://pokeapi.co/) 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.

## 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’s 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

All tests passed?

Great job!

You did it! Now you can take some rest.