Homework #7: Databases

In this homework, you will be populating a database table with information about Pokemon (we have provided the cache data in pokemon.json) and writing code to fetch data from the table that you create.

Pokemon, short for 'pocket monsters' (TIL), is a video game series from Nintendo with a corresponding card game, television show, and several movies. Pokemon are creatures with special abilities that can be caught and taught to battle by Pokemon Trainers. Each pokemon has a unique level of stats that affect their fighting ability, including hit points (HP), speed, attack power, and defense power. Many Pokemon can evolve into new forms from gaining battle experience. Every Pokemon has a type, some have more than one, that defines what type of abilities they can use (such as water, fire, or grass). Pokemon are most often caught by players to battle other Pokemon, but other activities exist for players and Pokemon to take part in such as fashion shows!

We have provided code for the following:

1. **read_data_from_file()**: To read the cache data in pokemon.json
2. **set_up_pokemon_database()**: To create an SQLite database and set up the 'connection' and 'cursor'
3. **set_up_types_table()**: To create/setup one of the SQLite tables for you. This table is called 'Types'. Run the starter code and then check the structure of the 'Types' table using the DB Browser!

These functions are provided for you, DO NOT CHANGE THE CODE.

When done with the assignment, your database will have two tables, including the one already provided for you. The other one, you will write the code for the create and fill!

We have also provided test cases that will pass if the functions are written correctly.
DO NOT EDIT the test cases in any way.

Tasks

1. **create_pokemon_table(data, cur, conn)**: This function takes 3 arguments as input:

    - the data (written in JSON format),
    - the database cursor,
    - the database connection object

    It iterates through the data present in pokemon.json and creates a table **Pokemon** with the following columns:

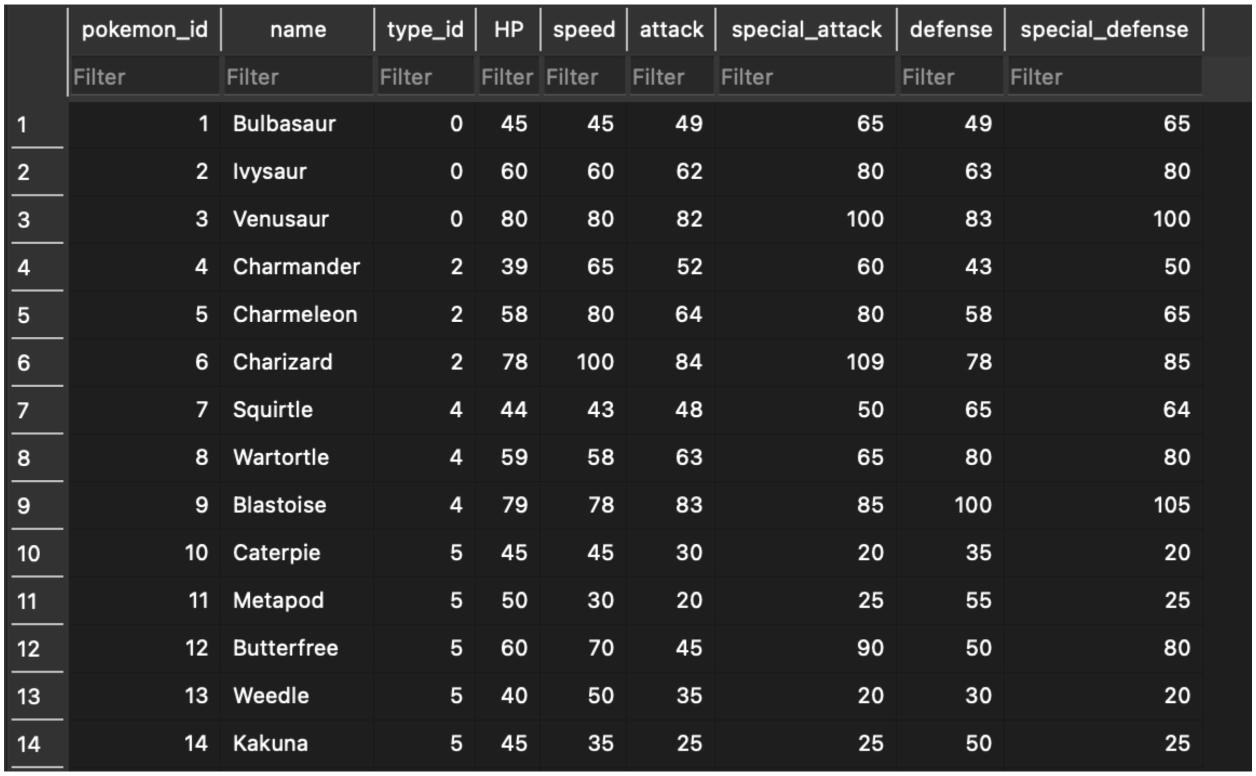
    - **pokemon_id** (datatype: integer and Primary key)
    - **name** (datatype: text); *Store the Pokemon's full English name*
    - **type_id** (datatype: integer)
    - **HP** (datatype: integer)
    - **speed** (datatype: integer)
    - **attack** (datatype: integer)
    - **special_attack** (datatype: integer)
    - **defense** (datatype: integer)
    - **special_defense** (datatype: integer)

    The function then loads all of the data present in pokemon.json into this table. It returns nothing.

    Note that some Pokemon may have two types. In these cases, use the first type for the type_id (e.g. ['Grass', 'Fire'] -> 'Grass' is the first type).

    See **set_up_types_table()** and the 'Types' table itself in the DB Browser to get a better understanding.

Expected table in DB Browser:



	pokemon_id	name	type_id	HP	speed	attack	special_attack	defense	special_defense
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Bulbasaur	0	45	45	49	65	49	65
2	2	Ivysaur	0	60	60	62	80	63	80
3	3	Venusaur	0	80	80	82	100	83	100
4	4	Charmander	2	39	65	52	60	43	50
5	5	Charmeleon	2	58	80	64	80	58	65
6	6	Charizard	2	78	100	84	109	78	85
7	7	Squirtle	4	44	43	48	50	65	64
8	8	Wartortle	4	59	58	63	65	80	80
9	9	Blastoise	4	79	78	83	85	100	105
10	10	Caterpie	5	45	45	30	20	35	20
11	11	Metapod	5	50	30	20	25	55	25
12	12	Butterfree	5	60	70	45	90	50	80
13	13	Weedle	5	40	50	35	20	30	20
14	14	Kakuna	5	45	35	25	25	50	25

2. **get_pokemon_by_spdefense_range(spdefense_min, spdefense_max, cur):** This function takes 3 arguments as input:

    - **spdefense_min: integer**
        - Passed in special defense MIN value
    - **spdefense_max: integer**
        - Passed in special defense MAX value
    - **cur: cursor**
        - The database cursor object

    It selects all Pokemon which have a **special defense value inside that range (inclusive)**.
    —

    Return a list of tuples, where each tuple contains the pokemon_id, name, and special defense.

    Expected output for **get_pokemon_by_spdefense_range(40, 50, cur):**

[(4, 'Charmander', 50), (17, 'Pidgeotto', 50), (25, 'Pikachu', 50), (29, 'Nidoran', 40), (32, 'Nidoran', 40)..., (104, 'Ducklett', 50)]

---

3. **get_top_attack_pokemon(base_attack, cur):** This function takes 2 arguments as input:

    - **base_attack: integer**
        - Passed in attack value
    - **cur: cursor**
        - The database cursor object

    A "good" attack value for a Pokemon is generally considered to be around 100 to 110. The function selects the top pokemons with a base attack value of at least *base_attack* (being passed into the function) and sorts them in descending order based on their attack value.
    —

    Return a sorted list of tuples based on the Pokemon’s attack value, in descending order, where each tuple contains the Pokemon’s name and attack.

    Expected output for **get_top_attack_pokemon(100, cur):**

    [('Archeops', 140), ('Kingler', 130), ('Rhydon', 130), ('Hitmonlee', 120)...,('Sandslash', 100)]

---

4. **get_pokemon_physical_sweepers(attack, speed, HP, cur):** This function takes 4 arguments as input:

    - **attack: integer**
        - Passed in attack value
    - **speed: integer**
        - Passed in speed value
    - **HP: integer**
        - Passed in HP value
    - **cur: cursor**
        - The database cursor object

    For the purpose of this function, a "physical sweeper" is defined as a Pokemon with an attack value greater than its speed and HP values.

It selects all the Pokemon at a specified attack value, speed greater than the value passed to the function, and HP greater than the value passed into the function.
—

Return a list of tuples, where each tuple contains the Pokemon’s name, attack, speed, and HP.

Expected output for **get_pokemon_physical_sweepers(60, 30, 20, cur):**

[('Pidgeotto', 60, 71, 63), ('Spearow', 60, 70, 40), ('Ekans', 60, 55, 35), ('Magneton', 60, 70, 50)]

---

5. **get_pokemon_type_above_speed_defense(pokemon_type, speed, defense, cur):** The function takes 4 arguments as input:

    - **pokemon_type: str**
        - Passed in type (e.g. 'Grass')
    - **speed: integer**
        - Passed in speed value
    - **defense: integer**
        - Passed in defense value
    - **cur: cursor**
        - The database cursor object

    It selects all the Pokemon that match the type (not type_id), have a speed greater than the value passed to the function, and a defense greater than the value passed to the function.
    —

    Return a list of tuples, where each tuple contains the Pokemon’s name, type (not type_id), speed, and defense.

    Note: You have to use a JOIN for this task. Remember, some Pokemon may have two types. In these cases, use the <u>first type</u> for the type (e.g. [‘Grass’, ‘Fire’] -> ‘Grass’ is the first type).

    Expected output for **get_pokemon_type_above_speed_defense("Ground", 40, 50, cur):**

    [('Sandslash', 'Ground', 65, 110), ('Marowak', 'Ground', 45, 110)]

Extra Credit (6 points)

get_pokemon_type_above_attack_HP(pokemon_type, attack, HP, cur): This function takes 4 arguments as input:

- **pokemon_type: str**
  - Passed in type (e.g. 'Grass')
- **attack: integer**
  - Passed in attack value
- **HP: integer**
  - Passed in HP value
- **cur: cursor**
  - The database cursor object

It selects all the Pokemon that match the type (not type_id), have an attack greater than the value passed to the function, and a HP greater than the value passed to the function.

Note: You have to use a JOIN for this task. Remember, some Pokemon may have two types. In these cases, use the first type for the type (e.g. [‘Grass’, ‘Fire’] -> ‘Grass’ is the first type).

Tasks

1. Return a list of tuples, where each tuple contains the Pokemon name, type (not type_id), attack, and HP.
2. Write **8 test cases** that successfully pass in test_pokemon_type_above_attack_HP to receive full points.
   - **Organize your tests into TWO sets of test cases.**
     1. **Each SET** should call get_pokemon_type_above_attack_HP() with a specific scenario in mind and consist of **4 test cases**.
        - e.g. Check for a Grass type Pokemon with an attack of at least 70 and HP of at least 80.
     2. Then, use the actual output from the function call for that specific scenario to:
        - Check that each tuple in the returned list contains the correct number of elements.
        - Verify that the correct Pokemon are being returned based on the given criteria.

Grading Rubric

1. **create_pokemon_table() - 15 Points**
    a. 10 points for creating all 9 columns in the table (using the correct data type for each column)
    b. 5 points for entering all 106 Pokemon in the table

2. **get_pokemon_by_spdefense_range() - 10 points**
    a. 5 points for correctly selecting the Pokemon based on stated criteria
    b. 5 points for returning the list of tuples with the pokemon_id, name, and special defense

3. **get_top_attack_pokemon() - 10 points**
    a. 5 for returning the list of tuples with the correct column values in each tuple
    b. 5 points for correctly outputting a list of tuples with the Pokemon name and attack

4. **get_pokemon_physical_sweepers() - 10 points**
    a. 5 points for correctly selecting the Pokemon based on stated criteria
    b. 5 points for returning the list of tuples with the Pokemon name, attack, speed, HP

5. **get_pokemon_type_above_speed_defense() - 15 points**
    a. 5 points for correctly using a JOIN
    b. 5 for returning the list of tuples with the correct column values in each tuple
    c. 5 points for correctly outputting a list of tuples with Pokemon name, pokemon_type, speed, and defense.

6. **EXTRA CREDIT + test_pokemon_type_above_attack_HP - 6 points**
    a. 4 points for correct implementation (using a JOIN, returning list of tuples with the correct column values)
    b. 2 points for adding eight test cases that successfully test the function