

Assignment 2

Python, PostgreSQL, psycopg2

Last updated: Friday 12th April 5:46pm

Most recent changes are shown in **red** ... older changes are shown in **brown**.

[Assignment Spec] [\[SQL Schema\]](#) [\[SQL Data\]](#) [\[Examples\]](#) [\[Testing\]](#) [\[Fixes+Updates\]](#)

Aims

This assignment aims to give you practice in

- implementing Python scripts to extract and display data from a database
- [optionally (but recommended)] implementing SQL views and PLpgSQL functions to support the scripts
- [optionally (but recommended)] implementing a collection of Python functions to support the scripts

You could complete this assignment with minimal use of SQL

But it is highly recommended that you use SQL for its intended purpose

Use SQL queries, views, and functions to filter and manipulate the data

Use Python to format and display the data

The goal is to build some useful data access operations on the Pokémon database.

Summary

Marks:	This assignment contributes 15 marks toward your total mark for this course.
Submission:	via WebCMS3 or <code>give</code> , submit the files <code>q1.py</code> , <code>q2.py</code> , <code>q4.py</code> , <code>q5.py</code> , <code>helpers.py</code> , <code>helpers.sql</code>
Deadline:	Monday 22 April 2023 @ 21:59:59
Late Penalty:	0.2 percent off the raw mark for each hour late, for 5 days Any submission after 5 days scores 0 marks This is the UNSW standard late penalty.

How to do this assignment:

- read this specification **carefully and completely**
- familiarise yourself with the [database schema](#)
- create a directory for this assignment
- copy the supplied files into this directory
- login to `vxdb2` and run your PostgreSQL server
- create a database `pkmon` on the `vxdb2`
- load the provided SQL dump file into the database
- explore the database

- complete the tasks below by editing q1.py, q2.py, q3.py, q4.py, q5.py
- test your work on vxdb2
- submit your python scripts via WebCMS or give (you can submit multiple times, only your last submission will be marked)

And, of course, if you have PostgreSQL installed on your home machine, you can do all of your development there.

But don't forget to test it on vxdb2 before submitting.

- helpers.sql ... any views or PLpgSQL functions to assist your Python
- helpers.py ... any Python function to share between scripts
- q1.py ... Python script to list the number of pokemon and the number of locations in each game
- q2.py ... Python script to list all possible encounters with a given Pokemon
- q3.py ...
- q4.py ...
- q5.py ... Python script to display the evolution chain of a given Pokemon

There are even some functions given in helpers.sql and helpers.py.
Freebie!

Background

Pokémon is a Japanese media franchise managed by The Pokémon Company, founded by Nintendo, Game Freak, and Creatures.

The franchise was created by Satoshi Tajiri in 1996, and is centered around fictional creatures called "Pokémon".

~ Wikipedia

Specifically for this assignment, we are interested in the Pokémon video games.

The Pokémon games are basically just databases with each game updating the User Interface.

~ Dylan Brotherston (describing this assignment to JAS), 2020

Pokémon have a lot of information associated with them.
And many relationships between different game elements.

Websites like [Bulbapedia](#), [The Pokémon Database](#), [Serebii](#), and even the official [Pokémon website](#) have searchable databases of Pokémon, moves, abilities, locations, and much more.

For this assignment (with a lot of python scripts and web scraping), we have set up a PostgreSQL database containing information about (almost) all 1008 Pokémon from all 9 generations, as well as all moves, abilities, and locations.

The Pokémon database for this assignment is not a database for a specific Pokémon game. Rather, it contains a large amount of general information

about Pokémon capabilities. If this database was combined with tables to hold the game state, then it would form a basis to run a specific Pokémon game. There is much more detail on what is in the database and what all the tables represent, in the "Database Design" page.

Setting Up

In addition to the database dump file, available on the [SQL Data](#) page, you are also provided a template Python files, and Python and SQL helper files.

The "template files" aim to save you some time in writing Python code. E.g. they do handle the command-line arguments and let you focus on the database interaction.

The `helpers.py` and `helpers.sql` files are provided in case you want to define Python functions or PLpgSQL functions that might be useful in several of your scripts. You are not required to use them (i.e. you can leave them unchanged).

The template files are available in a single [ZIP](#) or [TAR](#) file, which contains the following:

or copy them to your CSE account with the following command:

```
$ cp /web/cs3311/current/assignments/ass2/files/* .
```

Style

2 marks for the assignment will be based on the style of your code.

Similarly to the previous assignment, the main things to look out for are:

- readability
- consistency

above matching any specific style guide.

But in saying that, Python has an official style guide [PEP 8](#) that we recommend you follow (again not explicitly required, just a suggestion).

Python also has tools like [black](#), [autopep8](#), [pylint](#), that can identify and fix many of the common style issues.

Script Design

Python scripts should be designed with the following principles in mind:

- Use SQL to extract data in a form that is easy to process
- Use Python to take the data, format it, and produce output

In other words Data queries, filtering, grouping, sorting, etc should be done in SQL

while data formatting, conditions, error checking, etc should be done in Python.

You **should not** be pulling 1000s of rows from the database and then filtering them in Python

or matching foreign key values between separate queries in Python.

Such practices typically lead to inefficient code.

Python scripts that take longer than 2 seconds to execute will be penalised.

Python scripts that take longer than 5 will be killed and receive a mark of 0.

Your Python scripts shouldn't need more than half a second in the worst case.

And, of course, you should follow the normal abstraction practices you have learned in earlier programming courses, e.g. repeated sections of code should be placed in functions, etc.

Exercises

Q1 (3 Mark)

In the file `q1.py` write a script that takes 0 command line arguments, and prints information about the number of pokemon and the number of locations in each pokemon game.

Display each game on a separate line, using the following formatting:

```
f"{RegionName:<6} {GameName:<17} {#Pokemon:<8} {#Locations}"
```

Add a heading at the start of the output in the following formatting:

```
Region Game                #Pokemon #Locations
```

Your output should be ordered by the data in the order above.

That is: order first by Region, then by Game Name, etc.

Q2 (4 Marks)

In the file `q2.py` write a script that takes 1 command line argument:

1. The name of a pokemon

and prints all distinct locations where this Pokémon can be encountered.

Your output should include the following columns:

1. Game Name
2. Location Name
3. Encounter Rate (Rarity)
4. Minimum Encounter Level
5. Maximum Encounter Level
6. Encounter Requirements (as a comma separated list)

The Rarity of an encounter should be display as follows:

- "Common": 21% or higher
- "Uncommon": 6% to 20%
- "Rare": 1% to 5%
- "Limited": 0%

Your output should first be ordered by the Region name of each game, then by the data in the order above.

Your output should be formatted as a table, with columns all aligned.

Unlike Q1, which has a fixed width output, the width of the columns will be determined by the data.

So your program will need to dynamically determine the width of each column. examples of this formatting can be found on the examples page.

If your input is not a valid Pokémon name, print an error message as shown on the examples page, and exit.

Q4(6 Marks)

In the file `q4.py` write a script that takes 3 command line argument:

1. The name of a game in the pokemon database
2. The name of a pokemon that is attacking
3. The name of a pokemon that is defending

and prints all possible moves that the attacking pokemon can use against the defending pokemon.

Your output should include the information

- The name of the move
- The effective power of the move when used against the defending pokemon
- The requirements for the attacking pokemon to learn the move

The effective power of a move is calculated as follows:

The moves power from the `moves` table

multiplied by 1.5 (and rounded down) if the moves type is the same as either of the attacking pokemon's types

multiplied by the effectiveness of the moves type against the defending pokemon's type(s) (rounded down)

Eg. If Diglett was attacking Magnemite with Bulldoze

Bulldoze has a power of 60

Diglett has a first type of Ground, which matches Bulldoze's type, so the power is multiplied by 1.5 to get 90

Magnemite has a first type of Electric, which is weak to Ground, so the power is multiplied by 2 to get 180

Magnemite has a second type of Steel, which is weak to Ground, so the power is multiplied by 2 to get 360

So the effective power of bulldoze when used by Diglett against Magnemite is 360

Your output should first be ordered by the effective power of the move, then by the name of the move.

If A move can be learned by a pokemon in multiple ways, then the requirements should be ordered by the ID of the requirement.

The format of your output is shown on the examples page.

Q5 (7 Marks)

In the file `q5.py` write a script that takes 1 command line argument:

1. The name of a pokemon

and prints the evolution chain of that pokemon.

The evolution chain of a pokemon is the sequence of pokemon that a pokemon can both evolve from, and evolve into.

eg. Bulbasaur -> Ivysaur -> Venusaur

So if any one of these three pokemon names were given as input then all three would be printed.

But the order of the output would be different as we would be walking the chain in a different direction.

If "Bulbasaur" was given as input your script should first check if any pokemon evolve into "Bulbasaur". which they do not. stopping the chain (backwards).

Then check if "Bulbasaur" evolves into any other pokemon.

As "Bulbasaur" evolves into "Ivysaur", you should then check if "Ivysaur" evolves into any other pokemon.

As "Ivysaur" evolves into "Venusaur", you should then check if "Venusaur" evolves into any other pokemon.

Lastly you should check if "Venusaur" evolves into any other pokemon. which it does not. stopping the chain (forwards).

If "Venusaur" was given as input your script should first check if any pokemon evolve into "Venusaur".

As "Ivysaur" evolves into "Venusaur", you should then check if any pokemon evolve into "Ivysaur".

As "Bulbasaur" evolves into "Ivysaur", you should then check if any pokemon evolve into "Bulbasaur".

Then you should check if any pokemon evolve into "Bulbasaur". which they do not. stopping the chain (backwards).

Lastly you should check if "Venusaur" evolves into any other pokemon. which it does not. stopping the chain (forwards).

Both examples find the same chain, but the output order is different

Another different order could be found by starting at "Ivysaur".

Pre-evolutions (moving backwards) should be printed before post-evolutions (moving forwards) in the chain.

Not all evolutions are simple 1-to-1 evolutions.

Some evolution chains split.

If an evolution chain splits, then the script should print all possible evolutions.

Your script should also print the requirements for each evolution.

some evolutions have multiple requirements.

These requirements can be combined with an "and" or an "or".

ie, for X to evolve into Y, it must be A AND B, or it must be C OR D

An AND relationship is represented in the database as one row in the `evolutions` table, with multiple associated rows in the `evolution_requirements` table.

An OR relationship is represented in the database as multiple rows in the `evolutions` table, that have the same pre and post evolution pokemon.

Your output should be formatted as shown on the examples page.

in order to show the relationships between the evolution requirements.

The order of your output depends on the recursive algorithm you use to traverse the evolution chain.

So cannot be easily described.

But whenever you have a choice of two pokemon to print, you should print the one with the lowest ID first.

Similarly when ordering requirements, when you have a choice, you should print the one with the lowest ID first.

But additionally, non-inverted requirements should be printed before inverted requirements.

Examples

Examples of using these scripts can be found on the [Examples](#) page.