Assignment 2
Python, PostgreSQL, psycopg2

Last updated: **Friday 31st March 3:41pm**

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

[Assignment Spec] [Database Design] [Examples] [Testing] [Submitting] [Fixes+Updates]

Partial Release

This page is currently a partial release of the assignment.

The third question is now available

Two more questions will be released Soon^{TM}.

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 **16 marks** toward your total mark for this course.
Submission:	via WebCMS3 or `give`, submit the files `TBA`, `helpers.py`, `helpers.sql`
Deadline:	Friday 14 April 2023 @ 23:59:59
Late Penalty:	5% off the maximum achievable mark for each day late (i.e., 0.2% per hour) for the first 5 days, then 100% penalty after that

How to do this assignment:

- read this specification carefully and completely
- familiarise yourself with the database schema
- create a database `ass2` on the `vxdb2`
- load the provided SQL dump file into the database
- explore the database
- make a private directory for this assignment (e.g. `/home/$USER/COMP3311/ass2`. **not** under `/localstorage/$USER`)
- put a copy of the template files there
- edit the files in this directory on a host other than `vxdb2` (eg `vx05`)
- on `vxdb2`, test that your Python scripts produce the expected output

- submit the assignment via WebCMS3 or `give` (as described on the Submitting page)

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.

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.*` files are provided in case you want to defined 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), but they must still be submitted.

The template files are available in a single ZIP or TAR file, which contains the following:

- `helpers.sql` ... any views or PLpgSQL functions to assist your Python
- `helpers.py` ... any Python function to share between scripts
- `possible_evolutions` ... Python script to list pre and post evolutions of a Pokémon
- `attack_damage` ... Python script to find what move a Pokémon can use to deal the most damage against an opponent
- `pokemon_density` ... Python script to find what density of pokemon exist at each location
- `TBA` ... MORE TO COME

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, 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 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

To install the Pokémon database under your PostgreSQL server on `vxdb2`, simply run the following commands (after ensuring that your server is running):

```
$ dropdb ass2 # if it already exists
$ createdb ass2
$ psql ass2 -f /home/cs3311/web/23T1/assignments/ass2/dump.sql
```

We do not recommend that you copy the dump file to your own directory for this assignment.
The dump file is over 22MB in size and will take up a sizable amount of your disk quota.
For easy access to the dump file you may create a symbolic link to the dump file in the assignment directory.

```
$ ln -s /home/cs3311/web/23T1/assignments/ass2/dump.sql ass2.dump.sql
```

If you are working from home, where file size is probably not an issue, you can download the dump file dump.sql
It might take a minute or two to download.

If everything proceeds correctly, the load output should look something like:

```
SET
SET
...
SET
CREATE TYPE
... a few of these
CREATE DOMAIN
... a few of these
CREATE TABLE
... a few of these
COPY n
... a few of these
ALTER TABLE
... a whole bunch of these
```

You should get no ERROR messages.
The database loading should take less than 5 seconds on vxdb2, assuming that vxdb2 is not under heavy load.
(If you leave your assignment until the last minute, loading the database on vxdb2 may be considerably slower, thus delaying your work even more. The solution: at least load the database *Right Now*, even if you don't start using it for a while.)

Note that the database must be called ass2
(when you want to access it via your Python script).

A useful thing to do initially is to get a feeling for what data is actually there. This may help you understand the schema better, and will make the descriptions of the exercises easier to understand. Look at the schema. Run some queries. Do it now.

Examples:

```
$ psql ass2
... PostgreSQL welcome stuff ...
ass2=# \dt
... look at the list of tables ...
ass2=# \d pokemon
... examine the pokemon table ...
ass2=# SELECT Name, Average_Height, Average_Weight FROM Pokemon;
... look at the Pokemon table ...
ass2=# SELECT Pokemon.Name, Egg_Groups.Name FROM Pokemon JOIN In_Group ON In_Group.Pokemon = Pokemon.ID JOIN Egg_Groups ON In_
... look at Egg_Groups ...
ass2=# SELECT * FROM DBpop();
```

```
... how many records in all tables ...
ass2=# ... etc. etc. etc.
ass2=# \q
```

Style

6% of your mark (1 overall mark 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, do computation with 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

Possible Evolutions Script (5 Marks)

In the file `possible_evolutions` write a script that takes 1 command line argument:

1. the name of a Pokémon

and prints the Prevolutions and Evolutions of the given Pokémon.

Prevolutions are Pokémon that evolve into the given Pokémon.
Evolutions are Pokémon that the given Pokémon evolves into.

Your script should also print any requirements for the evolution to occur.

Pokémon are ordered by their National Pokédex number.
Requirements are ordered first by the Evolutions.ID followed by, Evolution_Requirements.Inverted and finally by, Requirements.ID

See the examples page for the exact output format.

Further Discussion on AND and OR requirements cna be found:

- Here on the forums
- Here on the forums

The sample solution takes ~0.05 seconds to run in the worst case.
Your should aim for no more than 0.2 seconds of execution time.

Pokemon Density Script (5 Marks)

In the file `pokemon_density` write a script that takes 1 command line argument:

1. the name of a region

And prints the name of each location in the given region, along with the average density of Pokémon in that location.

See the examples page for the exact output format.

"density" here literally means the physical property of density, which is the ratio of mass to volume.

Assuming that all Pokémon are perfect spheres, then the Average Height of a Pokemon is it's diameter.

Using the diameter of a Pokémon you can calculate the volume of a Pokémon.

```
volume = (4/3) * π * r^3
```

And using that volume along with the Average Weight of a Pokémon you can calculate the density of a Pokémon.

```
density = weight / volume
```

Note that the Average Weight of a Pokémon is in kilograms, and the Average Height is in metres.
But density is measured in grams per cubic centimetre. So you will need to convert the units accordingly.

As not all Pokémon are equally likely to appear in a location, you will need to take that into account.

You should scale the density of a Pokémon by it's rarity.

One you have found the density of a Pokémon in a location, find the sum of the densities of all Pokémon within the same game at that location.

And finally print the average of each of those sums.

For example:

```
There are 7 games in the Kalos region that have the location of "Route 2"
Those being:
        - Red
        - Blue
        - Yellow
        - Fire Red
        - Leaf Green
        - Let's GO, Pikachu
        - Let's GO, Eevee
for each of those games, find the sum of the scaled densities of all Pokémon that appear in that location
then find the average of those sums
repeat for all locations in the Kalos region
```

More Questions TBA

Attack Damage Script (10 Marks)

In the file `attack_damage` write a script that takes 3 command line arguments:

1. the name of a Pokémon (The attacking Pokémon)
2. the name of a second Pokémon (The defending Pokémon)
3. the name of a game in the Pokémon series

And prints a list of all the moves that the first Pokémon can use in the given game that will do damage to the second Pokémon.

When a Pokémon uses a move, it may deal damage anywhere within a range of possible damage values.

Your script should calculate minimum and maximum damage values for each move.
And print the name of the move, the minimum damage, and the maximum damage in order of the maximum damage.

See the examples page for the exact output format.

The damage formula is as follows:

$$\left(\frac{\left(\frac{2\times Level}{5}+2\right)\times Power\times A/D}{50}+2\right)\times random\times STAB\times Type$$

```
((((((2 * attacker-level) / 5) + 2) * attack-power * (attacker-attack / defender-defense)) / 50) + 2) * random-factor * STAB *
```

Where:

```
attacker-level is the level of the attacking Pokémon

attack-power is the power of the move being used

attacker-attack is the Attack (or Special Attack) stat of the attacking Pokémon (depending on the move category)

defender-defense is the Defense (or Special Defense) stat of the defending Pokémon (depending on the move category)

random-factor is a random number between 0.85 and 1.00

STAB is 1.5 if the attack is the same type as the attacking Pokémon, otherwise 1.0

type-effectiveness is the effectiveness of the attack against the defending Pokémon
```

Note:
To calculate minimum damage set the **attacker-level** to 1 and the **random-factor** to 0.85
To calculate maximum damage set the **attacker-level** to 100 and the **random-factor** to 1.00

As some Pokémon have two types:
If *either* of the attacking Pokémon's types is the same as the move's type, then the STAB is 1.5.

type-effectiveness starts with a value of 1.0
If *either* of the defending Pokémon's types are in the move's type effectiveness table, then the multiplier stored in the table should be applied.
If *both* of the defending Pokémon's types are in the move's type effectiveness table, then both multipliers should be applied multiplicatively.

```
e.g. if a move has base damage 100

If the Pokémon has two weaknesses then
100 * 2 * 2 = 400 damage.

If the Pokémon has two resistances then
100 * 1/2 * 1/2 = 25 damage.

If the Pokémon has one weakness and one resistance then
100 * 2 * 1/2 = 100 damage.
```

Note that "weaknesses" and "resistance" here are referring to rows within the Type_Effectiveness table.
A "weaknesses" being a multiplier of 200
A "resistance" being a multiplier of 50

The result of this formula is rounded down to the nearest integer.

Any move that does not include damage should be ignored.
Any move that deals 0 max damage after truncation of the result should be ignored.

The sample solution takes ~0.1 seconds to run in the worst case.
Your should aim for no more than 0.5 seconds of execution time.

Examples

Examples of using these scripts can be found on the Examples page.