1 Introduction to Embedded Databases and ORMs

In this lab we are going to look into embedded databases using **sqlite** and how to interact with it using native programming types and objects.

Clone the git repository, e.g., using the command—

```
git clone https://altitude.otago.ac.nz/cosc430/embedded-intro/
```

If you've forgotten how to do this, refer to the previous labs, or ask one of the teaching staff.

1.1 Use of Vagrant as a Linux environment inside which to do the lab

If your computer is running Linux or macOS, you may be able to do the lab directly on your normal operating system. Otherwise, you can instead perform the lab within a controlled Linux environment by using Vagrant. This is similar to how we ran CoreOS to support Docker containers, so you can refer to past labs for more information about Vagrant. The `Vagrantfile` required to start a Linux environment, including installing the Peewee ORM discussed below, is included as part of the repository that you have cloned for this lab.

Once you have successfully run `vagrant up` and `vagrant ssh` remember to `cd /vagrant` to change your shell's working directory in your virtual machine to a folder that is shared with your host computer, and contains the files cloned from this lab's git repository.

1.2 Sqlite Introduction

`sqlite` is a small (semi-)relational database system that supports ACID properties. It's lightweight and is used in all modern smartphone operating systems (e.g., iOS and Android), across desktop operating systems (macOS, Windows, Linux, *BSD, etc.) and within a large number of commercial and open source software systems (e.g., Firefox, Dropbox, etc.). Due to its widespread use, it is almost certainly the world's most popular database in terms of number of database instances, and is estimated to be the second most deployed piece of software in the world.

The entire database is stored in a single file and can easily be copied between computers with different architectures (e.g., 32-bit and 64-bit). For a complete list of features see: https://www.sqlite.org/about.html, while there are plenty of examples of high-profile users https://sqlite.org/famous.html (also in quite a literal sense of the word "high"—it's found its way onto the AirBus A350).

The aim of the following little exercise is to get you interacting with the sqlite command line tools (however there are GUI tools available, e.g., SQLite Browser and Jetbrain's DataGrip).

1.2.1 Navigating an Existing Database

When you cloned the repository you will have received a copy of the Spotify rankings (`nz-spotify.sqlite`). Paul retrieved this dataset from https://www.kaggle.com/edumucelli/spotifys-worldwide-daily-song-ranking. For our purposes, Paul extracted all the NZ rankings (the original size was ~300MB).

Run the following command to start an `sqlite` session:

```
sqlite3 nz-spotify.sqlite
```

You should see something resembling the following

```
SQLite version 3.26.0 2018-12-01 12:34:55
Enter ".help" for usage hints.
sqlite>
```

Have a look at the help by typing `.help` and you should see that there are various commands for interacting with the sqlite database. The first task is to answer the following questions (remember that sqlite supports many relational database features, including much of the SQL language):

- What is the name of the (only table)?
- What is the schema of the table?
- What is the date range of the data?
- Who has the most popular song over the period?
- Who has the least popular song over the period?
- What was the most popular song, and who sang it, over Christmas in the period?

Once you've finished with the database, close the `sqlite` session (either type `.exit`, or ctrl-d).

We will skip the table creation phase as it is something that you should all be familiar with—most normal SQL constructs should work without a problem (excepting any SQL dialect differences). The key idea is that each database is contained in a separate file—unlike other systems where you can switch databases, e.g., such as MySQL's (non-SQL) `USE database` command.

1.3 (A Brief) Python Introduction

We will jump straight into the deep end in terms of Python. The following code llisting defines a class of type `Person`.

```python
#!/usr/bin/env python3

class Person(object):
    """
    Define a simple 'Person' object with first and last names
    as well as an email address.
    """

    def __init__(self, first_name, last_name, email):
        """Constructs a new Person object."""

        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.phone = None    # This is Python's null value

    def setPhone(self, value):
        self.phone = value

    def getPhone(self):
        return self.phone

    def __str__(self):
        """Returns a String representation of the Person object."""

        return "{} {} [{}]".format(self.first_name,
            self.last_name, self.email)

david = Person("David", "Eyers", "dme@cs.otago.ac.nz")
print(david)   # you should see: David Eyers [dme@cs.otago.ac.nz]
```

In the `Person` class we have two functions, an `init` and a `str`. The `init` is the object's constructor, and the `str` behaves like Java's `toString` method. Other things to note are the documentation comment lines start and end with three double quotes—they must go below the function/class description (note that you are not expected to comment your code like this if you are just working through the labs, but this is shown here as an example of good practice), and finally the line comments start with `#`. The first line (starting with `#!`) is special and will be explained shorty.

Note that all the functions here accept `self`—this is equivalent to Java's `this`—and refers to the current object. So, when the `david` object talks about `self` it means itself, whereas the `haibo` object would refer to the `haibo` instance.

One last thing to note about Python is that spacing within the source files *is* important. Both the file provided and the example above use `tabs` to indent the lines—you can use spaces if you want, but you *must* be consistent throughout your code, or the Python interpreter may refuse to touch it.

When you cloned this repository you should also have received a copy of the above program. You can run it in one of two ways. Firstly, by typing `python3 person.py` or `./person.py`. In the first method, we're being explicit about which Python command we want to use to run the script. In the second form, we leave it up to the operating system to figure out which Python command to run —this is where the `#!` comes into play. That line, called the 'shebang' line, is saying 'use the following interpreter to run the script that follows'. In this case it would be the same as if we'd explicitly said to run `python3`, since that is what is referred to on the shebang line.

Complete the following tasks:

- Create a new instance using your details.
- Print your details above David's
- Add accessors and mutators (or getters and setters) for the other fields.
- Set David's phone number to `+64-3-479-5749`
- Add an `room` field and set it to `125, Owheo Building` for David
- Add these extra fields to the `str` function and ensure that you can see them when you run your script.

Hint: Add these changes one by one and run your program to test each such change—it's typically tricky to get the code right the very first time.

Now, from here there's a lot that can be done to extend this example, but we don't want to introduce unnecessary complexity that would distract from the point of this exercise, which is using the Object-Relational Mapper (ORM) to put data into and get data back from the database.

1.4 Object-Relational Mapping (ORM)

So, now that we've created some objects, let's assume we want to have them available for use later on, in another program. There are multiple ways to interact with the database programmatically. You can use database drivers and write SQL statements and queries directly, alternatively you can use an Object-Relational Mapping tool.

Python supports both these approaches, however in this lab we will explore the ORM route using a tool called `peewee`. We first need to install it. (On the Vagrant Linux system you should first run the command `export LC_ALL="en_US.UTF-8"`.) Run `pip3 install --user peewee` and you should see something resembling:

```
Collecting peewee
  Downloading peewee-3.2.2.tar.gz (592kB)
    100% |████████████████████████████████| 593kB 901kB/s
Installing collected packages: peewee
Installing collected packages: peewee
  Running setup.py install for peewee ... done
Successfully installed peewee
```

1.4.1 Creating a Model

Make a copy of the `person.py` can call it `person-db.py` (command: `cp person.py person-db.py`). We need to make some changes to the script so that it reads as follows (I've omitted the documentation for brevity):

```python
#!/usr/bin/env python3

from peewee import *

db = SqliteDatabase('person.sqlite')

class Person(Model):
    first_name = CharField()
    last_name = CharField()
    email = CharField()
    # add any other fields as appropriate

    class Meta:
        database = db

    # the __init__ function is removed
    # you'll have other accessors/mutators here: you can leave them

    def __str__(self):
        return "{} {} [{}]".format(self.first_name,
            self.last_name, self.email)

db.connect()
db.create_tables([Person])

david = Person(first_name = "David", last_name = "Eyers",
    email="dme@cs.otago.ac.nz")
print(david)    # you should see: David Eyers [dme@cs.otago.ac.nz]

david.save()

db.close()
```

Note that there are several difference between this program and the one above:

1. we're importing the `peewee` library;
2. we're creating a `SqliteDatabase` instance (called 'person.sqlite');
3. instead of `object` we now have a `Model` (this is Python's inheritance);
4. we've moved the declaration of our fields into the main part of the class;
5. there is now **NO CONSTRUCTOR**;
6. there is a new inner class—this is used by `peewee` to determine which database to use.

Most of those changes should be relatively straight forward—if you don't follow, ask one of the teaching team to explain what's going on. One of the details of Python we've skipped over is the syntax for lists. They make a (brief) appearance in `create_tables`—we can give the function multiple tables to make, thus we pass the function a list. This is indicated by the square brackets.

Run the program, and using the skills that you developed when working through the SQLite section above, have a look at the `person.sqlite` database. You should see that there is a single table, with columns matching the fields of the object and that there are some instances of the object stored in the database (if you only see one you've probably only saved that instance).

In your Python script add some more instances of `Person` and save them to the database. You may note that each time you run this program the same instance is created and saved to the database—to avoid this, you can remove the `person.sqlite` file (`rm person.sqlite`) if you like.

One small thing to note is that if you want to modify an instance you'll need to save it again. For example:

```
david = Person(first_name = "David", last_name = "Eyers",
    email="dme@cs.otago.ac.nz")
print(david)
david.save()

david.setPhone("+64-3-479-0000")
```

```
# this will not be stored in the database until you run:
david.save()
```

You can also remove items from the database in Python too:

```
# as above
david.delete_instance()
```

1.5 Querying model instances

Now that we've got some data stored in the database, we need to retrieve it. At the end of your `person-db.py` script there will be some lines that create instances of `Person` and save them to the database. We don't need them any more (unless you've deleted the database!), so comment them out:

```
#!/usr/bin/env python3

from peewee import *

db = SqliteDatabase('person.sqlite')

class Person(Model):

    # as above -- omitted for brevity

db.connect()

#db.create_tables([Person])
#
#david = Person(first_name = "David", last_name = "Eyers",
#    email="dme@cs.otago.ac.nz")
#
#print(david)        # you should see: David Eyers [dme@cs.otago.ac.nz]
#
#david.save()

# add the new lines here...

db.close()
```

Below these (newly) commented lines, add the following:

```
david = Person.select().where(
    Person.email == 'dme@cs.otago.ac.nz').get()
print(david.first_name)

for person in Person.select():
    print(person)
```

You should first the David printed then all the other people stored in the database.

The next task is to get you familiar with writing some data processing in Python. For that we need to have some more sample data. Add the following to your Python script and run it.

```
Person.create(first_name="Haibo", last_name="Zhang",
    email="haibo@cs.otago.ac.nz")
Person.create(first_name="Cathy", last_name="Chandra",
    email="cathy@cs.otago.ac.nz")
Person.create(first_name="Michael", last_name="Albert",
    email="malbert@cs.otago.ac.nz")
Person.create(first_name="Steven", last_name="Mills",
    email="steven@cs.otago.ac.nz")
Person.create(first_name="Andrew", last_name="Trotman",
    email="andrew@cs.otago.ac.nz")
```

Consulting the peewee documentation http://docs.peewee-orm.com/en/latest/, write some code that will:

- Count the number of people in the database
- Find the person with the shortest email address
- List anyone with an ‘e’ in their first name
- List anyone with an ‘e’ or ‘E’ in their last name
- Find the average length of first names

Note that while *these* tasks could easily be done in SQL, there are some computations that SQL cannot do so easily. Python has a large range of libraries available, including some data visualisations (e.g. matplotlib https://matplotlib.org/gallery/index.html), data analysis (e.g. pandas https://pandas.pydata.org/index.html), data modelling (scikit-learn http://scikit-learn.org/stable/), interactive lab notebooks (e.g jupyter http://jupyter.org) amongst plenty of others.

1.6 Interacting with Existing Databases

So the above is all well and good if you're creating a new database. What if we want to interact with an existing one? Well, `peewee` can handle that too. We'll use the Spotify dataset from the previous exercise and create a Python class to allow us to interact with the data. Run the command below:

```
./pwiz.py -e sqlite nz-spotify.sqlite > spotify.py
```

The `pwiz.py` is a script provided by the developers of `peewee` to read the database and generate our class from the schema of the table. The `-e sqlite` parameter is telling it which database engine to use (e.g., MySQL and PostgreSQL are supported too) and the name of the database is given as `nz-spotify.sqlite`. The final part of the command `> spotify.py` uses the command shell's file redirection capabilities to take the output of the `pwiz` command and write it to a file (this is quite a useful trick to know about).

Open the `spotify.py` file and you should be familiar with most of the contents—it should follow the example above fairly closely. At the end of the file, write some Python code to answer the queries below. You'll find that if you try to run the program using `./spotify.py` it'll complain at you about 'permission denied', type `chmod +x spotify.py` (to grant the executable permissions) and you should be ok.

- What is the date range of the data?
- Who has the most popular song over the period?
- Who has the least popular song over the period?
- What was the most popular song, and who sang it, over Christmas in the period?

2 Useful websites and reference documentation

Datasets:

- https://www.kaggle.com/
- Spotify Daily Song Ranking https://www.kaggle.com/edumucelli/spotifys-worldwide-daily-song-ranking

SQLite:

- Home page https://sqlite.org
- Documentation https://www.sqlite.org/docs.html

GUI Tools:

- SQLite Browser http://sqlitebrowser.org
- Jetbrain's DataGrip https://www.jetbrains.com/datagrip/

Peewee specific:

- Home page http://peewee-orm.com
- Documentation http://docs.peewee-orm.com/en/latest/
  - Quickstart http://docs.peewee-orm.com/en/latest/peewee/quickstart.html

Some useful Python libraries:

- matplotlib https://matplotlib.org/gallery/index.html
- pandas https://pandas.pydata.org/index.html
- scikit-learn http://scikit-learn.org/stable/
- jupyter http://jupyter.org

If you want to learn more about Python there are plenty of guides on the internet.

- https://www.datacamp.com/courses/intro-to-python-for-data-science
- https://github.com/jakevdp/PythonDataScienceHandbook
  - https://jakevdp.github.io/PythonDataScienceHandbook/—the whole book available for free