**SQLAlchemy** is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL. It is an open source ORM for Python with a slew of features. The database toolkit for python.

Below is the step to write a single Python file to set up and configure our database: **database\_setup.py** using SQLAlchemy

***Location of database\_setup.py:***

**Google drive \ICT Sharefolder\ICT Design\Implementation Teams\Backend Group\Database design\SQLAlchemy folder**

***Creating a database with SQLAlchemy has four major coding components.***

configuration

class

table

mapper

NOTE:

**configuration** code, which is what we use to import all the necessary modules.

**class** code that we use to represent our data in Python.

**table** that represents the specific table in our database.

**mapper** that connects the columns of our table to the class that represents it.

**####Writing the configuration code####**

* Configuration code sets up all of the dependencies needed for my database and binds the code to the SQLAlchemy engine.
* Create an instance of a class called declarative base, in order for our class to inherit all the features of SQLAlchemy.
* Add some configuration code at the end of our Python file to connect to an existing database. Or in this case, create a new one.

**####Writing the class and table code####**

* Class code is the object-oriented representation of a table in our database.
* Create a class name, and make it extend from the Base class we just created in the configuration code.
* Inside of this class declaration, Add all of the code for our table and mapper code.
* Create classes to correspond with the tables we want to create in the database
* Inside each of our classes, Create a table representation, wse the special variable, double underscore table name double underscore to let SQLAlchemy know the variable that we will use to refer to our table.
* To let SQLAlchemy know the variable that we will use to refer to our table.

**####Writing the mapper code####**

mapper code creates variables that we will use to create **columns** within our database.

When we create a column, we must also pass an attribute to that column.

**Executing of database\_setup.py**

When you run the **database\_setup.py ,** it will create an empty database with the following TABLE:

**"OrderDish”**

**"MenuDish"**

**"Table”**

**"UserAcc"**

**"Staff"**

**"Order"**

**####To populate the tables with information within a Python script####**

#We must first import the following dependencies from SQLAlchemy and the empty database we created, into our environment from our database setup.py file that we created

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker

#Import our base restaurant, and menu classes.

from database\_setup import Base, OrderDish, MenuDish, Table, Staff, Order

#The create\_engine function lets our program know which database engine we want to communicate with.

engine = create\_engine('sqlite:///restaurantmenue.db')

#Bind the engine to the base:

‘’’ This Bind command just makes the connections between our class definitions and the corresponding tables within our database.’’’

Base.metadata.bind = engine

#Create a sessionmaker object.

‘’’sessionmaker object establishes a link of communication between our code executions and the engine we just created.

In order to create, read, update or delete information on our database, SQLAlchemy executes database operations via an interface. It's called a session. A session allows us to write down all the commands we want to execute, but not send them to the database until we call a commit. ‘’’

DBSession = sessionmaker (bind = engine)

# create an instance of a DBSession and call it session for short.

‘’’ From now on, when I want to make a change to my database, I can do it just by calling a method from within session.

The DBSession object gives me a staging zone for all of the objects loaded into a database session object.

Any change made to the objects in the session won't be persisted into the database until I call session.commit.’’’

session = DBSession()

**####To add DATA to the Database: ####**

# Make new entry in the database

newEntry = ClassName(property=”value”,…)

customer = UserAcc (FirstName=”Wallace”, … )

session.add(newEntry)

session.add(customer)

session.commit()

**####To Query the Database####**

‘’’ Create a variable called result and set it equal to session.query(UserAcc) and use .first this time,

I now have a variable that corresponds to a single row in my database.

These single row references allow me to extract column entries as method names. So if I were to write firstResult.name, I'm given the name of my UserAcc object as it is stored in the database.’’’

session.query(UserAcc).all()

result = session.query(UserAcc).first()

result.name

Ref: <http://docs.sqlalchemy.org/en/rel_0_9/orm/query.html>

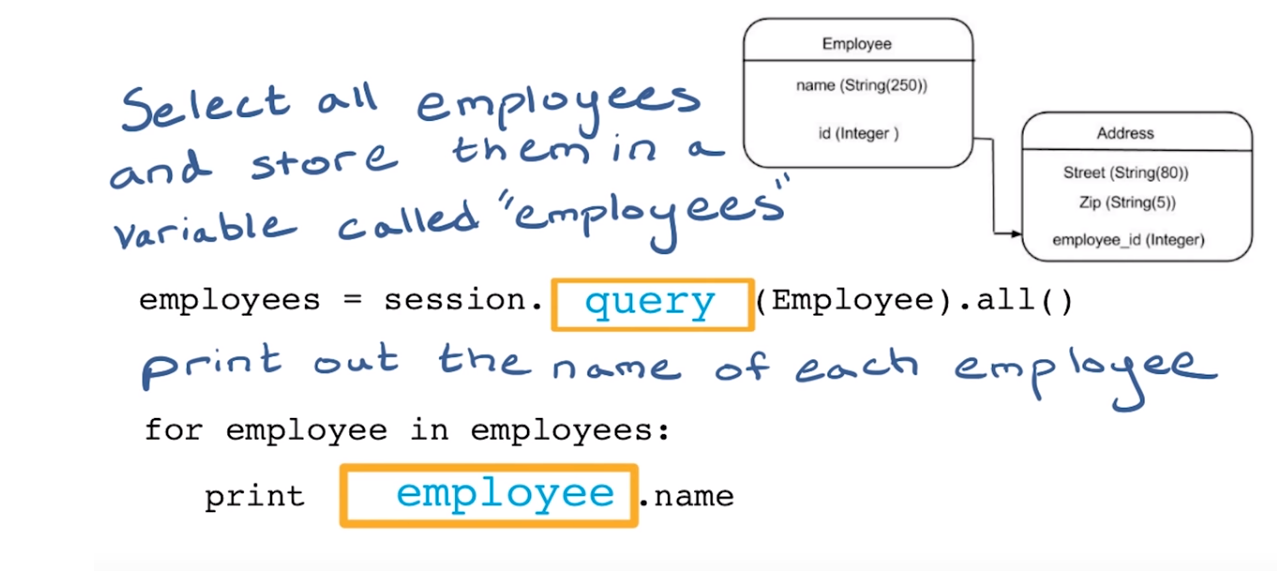
**Option2: use Python for loop to return the data that we want to see**.

#make a variable called items and set it equal to a query that retrieves all of my menu items.

items = session.query(UserAcc).all()

for item in items:

print item.name

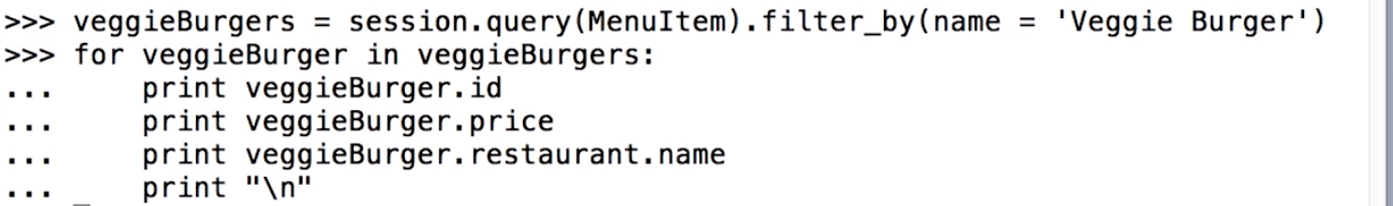


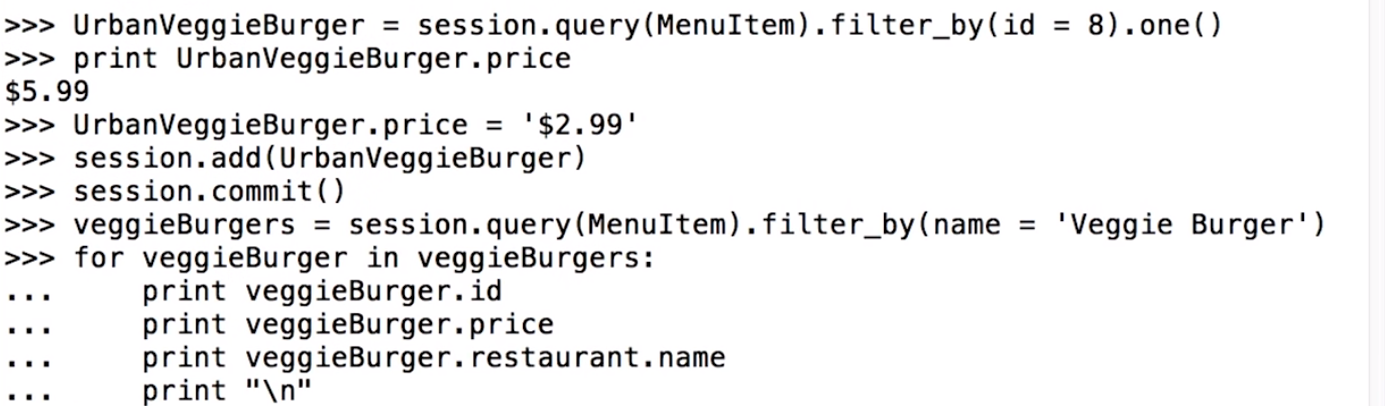
SQL Alchemy has an extensive collection of queries we can perform on DB:<http://docs.sqlalchemy.org/en/rel_0_9/orm/query.html>

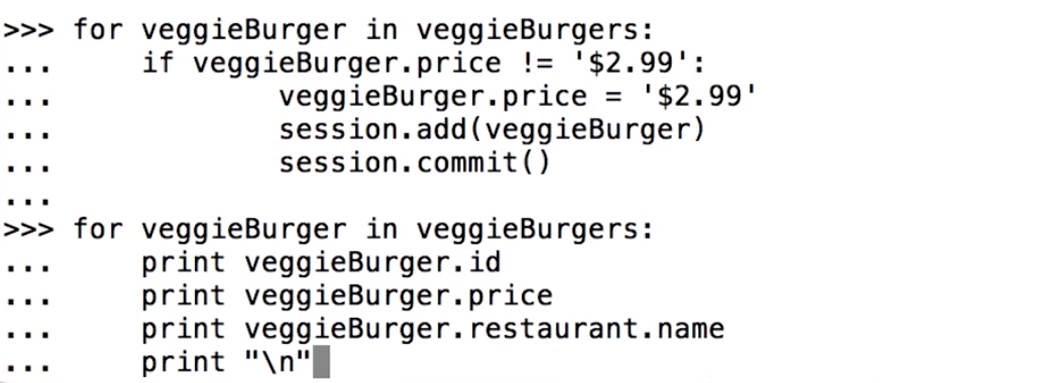
**####To Update the Database####**

1. Execute a query to find the item we want and store it in a variable.
2. declare the new price of the variable.
3. Add the variable to our session.
4. Commit the session to the database.

Use the filter\_by feature that lets us filter queries based on attribute entries:

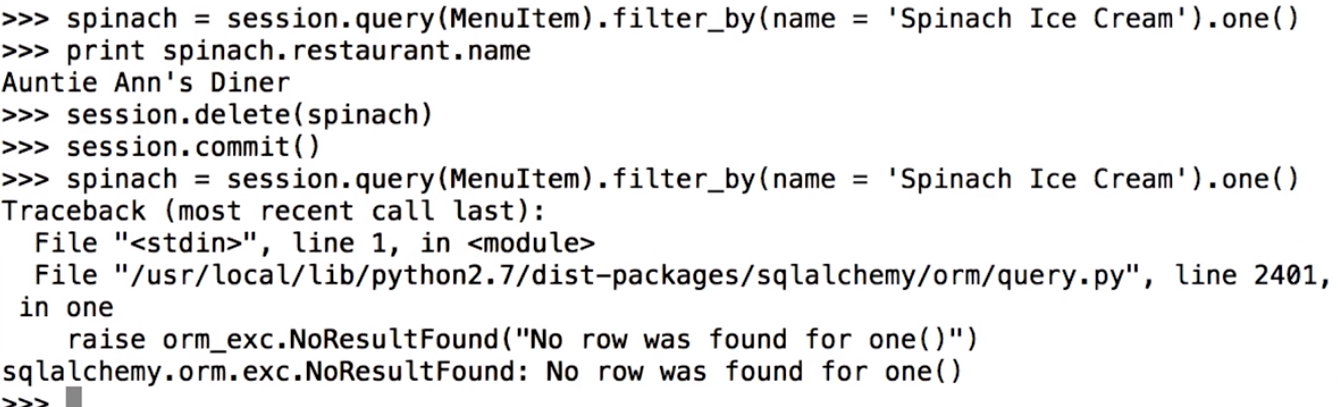






**####To Delete the Database####**

1. Execute the query to find the item
2. Store it in a variable.
3. Call session delete on that item.
4. Commit the session.



Ref:

<http://www.sqlalchemy.org/library.html#tutorials>

Start watching: <https://www.youtube.com/watch?time_continue=230&v=woKYyhLCcnU>