# SQL with Python

Phillip Luong
University of Hawaii – West Oahu

## Overview

In the coding and computer world people are familiar with Python and SQL. Python is a general coding language used to problem solve and create programs. SQL is a specific language used to store and retrieve specific information from databases. Then you have SQLAlchemy which is the bridge between both languages. This presents a method to use Python to create databases and facilitates the communication between Python programs and the databases. Object Relational Mapper (ORM) tool translates Python classes to tables on relational databases and automatically converts function calls to SQL statements.

## The Python and SQL Language

- Python is a general-purpose coding language which means that, unlike HTML, CSS, and JavaScript, it can be used for other types of programming and software development besides web development. That includes back end development, software development, data science and writing system scripts among other things.

- SQL stands for standard query language, it is used to communicate with a database. It is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.

## What is SQLAlchemy?

- SQLAlchemy is a well-regarded database toolkit and object-relational mapper(ORM) implementation written in Python. SQLAlchemy provides a generalized interface for creating and executing database-agnostic code without needing to write SQL statements.

3 most imprtant components when writing a SQLAlchemy code:

1. A Table that represents a table in a database.
2. A mapper that maps a Python class to a table in a database.
3. A class object that defines how a database record maps to a normal Python object.

- Instead of having to write code for Table, mapper and the class object at different places, SQLAlchemy's declarative allows a Table, a mapper and a class object to be defined at once in one class definition.

## Python + SQL = SQLAlchemy

```
"""\
Creating a example database using SQLAlchemy
"""
# We use the first couple of lines to declare a mapping called the configurational process.
from sqlalchemy import Column, String, Integer, Float, ForeignKey, Table
from sqlalchemy.orm import relationship, backref
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy import select

Base = declarative_base()
# Now that we have a "base", we can define any number of mapped classes in terms of it.  Example of DDL
class MODEL(Base):
   __tablename__ = 'MODEL'
# Here we define columns for the table person
   Mod_Id = Column(Integer, primary_key = True)
   Mod_Name = Column(String)
   Mod_Seats = Column(Integer)
# Create an engine that stores data in the local directory's
engine = create_engine(DBRESOURCE)
Base.metadata.create_all(engine)
# Create all tables in the engine. This is equivalent to "Create Table" statements in raw SQL. Example of DML
conn.execute(table.insert(),[
  {'Mod_name':'Foxtrot','Mod_seats':'4'},
  {'Mod_name':'Tango','Mod_seats':'2'}])
# Insert multiple data into the "model" table, you can do it this way or assign a list, either way works.
# Now that you created a basic database, you can start querying or pulling the data out. There are different
# ways which you can get the data you want. Example of  DQL Method 1:
# find_it = select([Department.id])
sel_rows = select([Model])
# rs = s.execute(find_it)
rs = s.execute(sel_rows)
# print(rs)
for r in rs:
   print( "Row:", r )
# select * from 'Model' Method 2:
select_st = select([table]).where(
  table.c.l_name == 'MODEL')
res = conn.execute(select_st)
for _row in res:
   print(_row)
```

Output

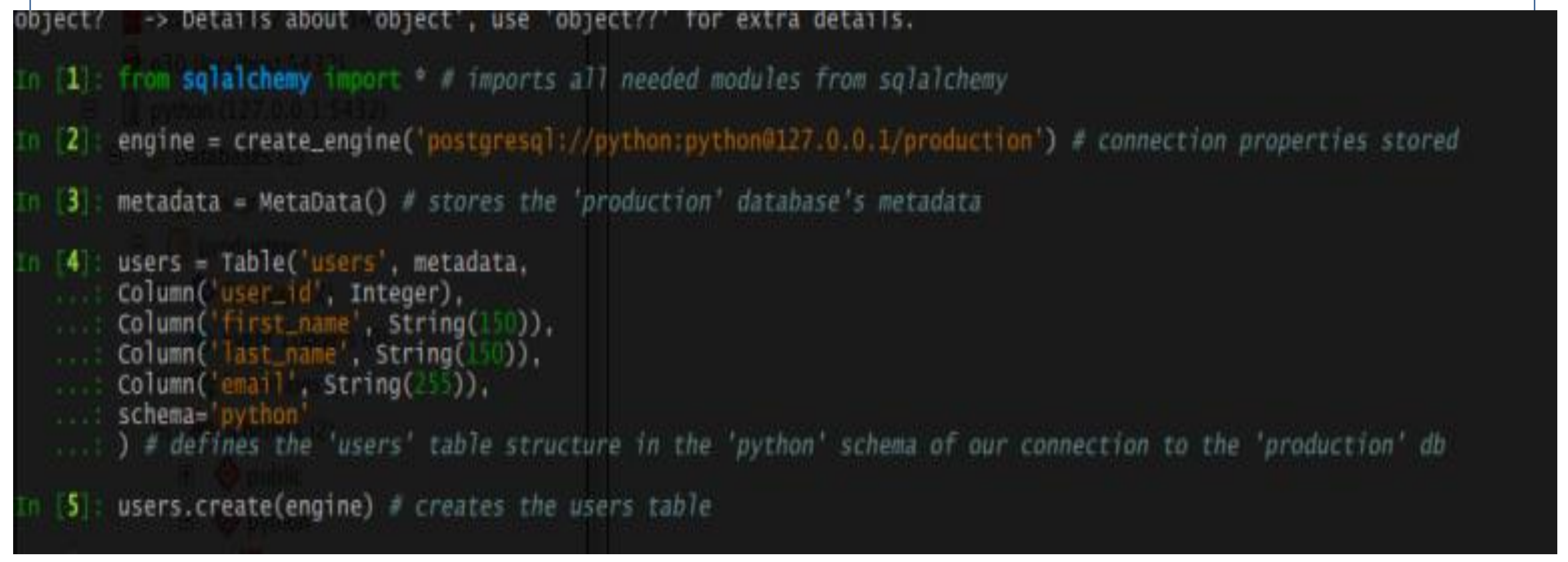
```
(1, 'Foxtrot', 4)
(2, 'Tango', 2)
```

## Why use SQLAlchemy?

- SQLAlchemy isn't just an ORM, it also provides SQLAlchemy Core for performing database work that is abstracted from the implementation differences between PostgreSQL, SQLite, etc. In some ways, the ORM is a bonus to Core that automates commonly-required create, read, update and delete operations.
- A benefit many developers enjoy with SQLAlchemy is that it allows them to write Python code in their project to map from the database schema to the applications' Python objects. No SQL is required to create, maintain and query the database. The mapping allows SQLAlchemy to handle the underlying database so developers can work with their Python objects instead of writing bridge code to get data in and out of relational tables.

## Conclusions

Therefore, SQLAlchemy makes creating databases with python easy. The language may seem confusing at first but once you understand the concept of this ORM, it'll be easy sailing. Of course you have to know the basics of python and SQL first before SQLAlchemy. SQLAlchemy is great because it provides a good connection / pooling infrastructure; a good Pythonic query building infrastructure; and then a good ORM infrastructure that is capable of complex queries and mappings (as well as some pretty stone-simple ones).



```
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: from sqlalchemy import * # imports all needed modules from sqlalchemy

In [2]: engine = create_engine('postgresql://python:python@127.0.0.1/production') # connection properties stored

In [3]: metadata = MetaData() # stores the 'production' database's metadata

In [4]: users = Table('users', metadata,
   ...: Column('user_id', Integer),
   ...: Column('first_name', String(150)),
   ...: Column('last_name', String(150)),
   ...: Column('email', String(255)),
   ...: schema='python'
   ...: ) # defines the 'users' table structure in the 'python' schema of our connection to the 'production' db

In [5]: users.create(engine) # creates the users table
```

## Contact

Phillip Luong

University of Hawaiʻi–West Oʻahu
Kapolei, HI  96707

## References

1. Adam Lambert, *Fundamentals of Python: First Programs | 2nd Edition,* 2019
2. Carlos Coronel and Steven Morris, *Database Systems: Design, Implementation, & Management | 13th Edition,* JAN 2018
3. SQLAlchemy authors and contributors, https://www.sqlalchemy.org/
4. Xiaonuo Gantan, https://www.pythoncentral.io/introductory-tutorial-python-sqlalchemy/ , APR 2013
5. https://github.com/ics111-204/c21-luongp