Python and SQLAlchemy

DATABASE ACCESS AND OBJECT-RELATIONAL MAPPING

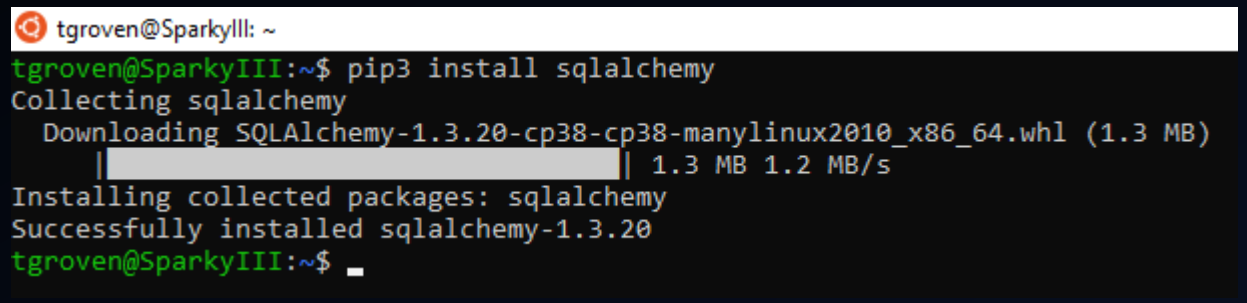
Presented By: Tim Groven

Topics Covered

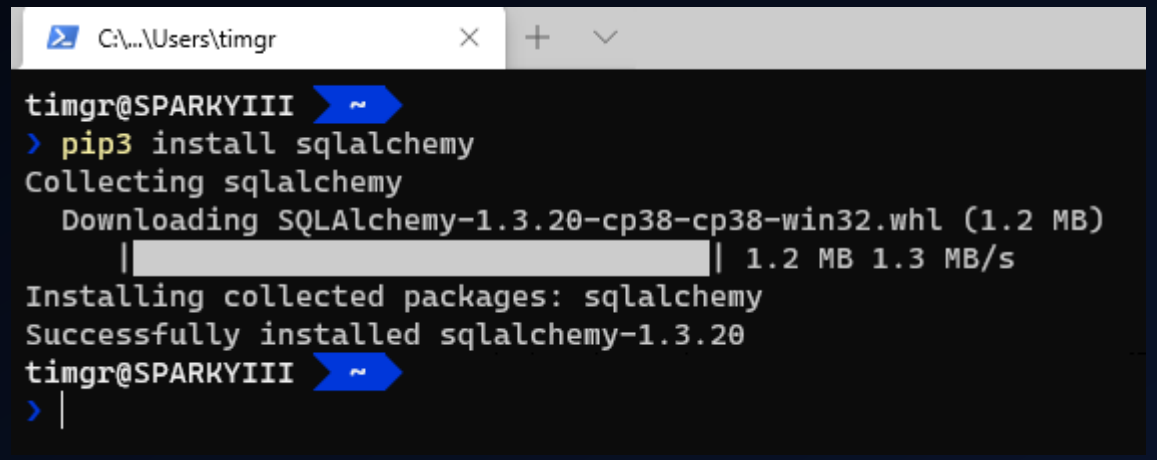
- Installation
- AdventureWorks
- Major Components
- Examples
- Object-Relational Mapping

Installation - SQLAlchemy

- Ubuntu – pip3 install sqlalchemy



- Windows – pip3 install sqlalchemy



ODBC Driver (Dialect)

- DBAPI Interface
- Many to choose from
  - PyODBC – Open Source, Used for this demonstration
  - mxODBC – Pay per License
  - Pymssql – No longer maintained, lacks new features
  - zxJDBC for Jython – Experimental, Java/Python hybrid
  - Adodbapi – Not supported in SQLAlchemy 0.6 and above (Currently 1.3)

Installation - pyodbc

- Ubuntu
  - sudo apt install unixodbc-dev
  - pip3 install –user pyodbc
- Windows
  - pip3 install pyodbc

AdventureWorks

For this presentation, we will be using Microsoft's AdventureWorks Lightweight sample database.

This database is free, and can be downloaded here:

https://docs.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver15&tabs=ssms

Major Components - Connection String

Format:

“{dialect}+{driver}://{username}:{password}@{server}:{port}/{databasename}?driver=ODBC+Driver+17+for+SQL+Server”

For these examples:

“mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server”

Major Components - Engine

This is the base from which all your database access comes from.

Create an instance:

```
engine = create_engine(
    "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server")
```

No connection to the database is done here.

Major Components - Engine

Engine creation arguments (not all):

- echo – Logs all statements to the default log handler. Default is false.
- Encoding – Denotes the language encoding to use with the database. Default is utf-8.
- Isolation_level – Dialect interpreted for the transaction isolation level of the connection. Default is dialect dependent.

Major Components - Engine

Passing arguments to the engine uses basic Python keyward arguments.

```
engine = create_engine(
    "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server",
    encoding='latin1', echo=True)
```

Major Components - Connection

This is what provides high-level functionality for a wrapped DB-API connection.

Create an instance:

```
engine = create_engine(
    "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server")
conn = engine.connect()
```

Major Components - Connection

Common methods:

- begin() – Creates a Transaction object.
- close() – Returns the connection back to the pool.
- execute() – Executes a SQL statement and returns a ResultProxy object.

Major Components - Transaction

Encapsulates all SQL commands inside a transaction that is committed once the code block completes.

If an exception is raised, the transaction is rolled back, and the exception is sent outwards.

```
engine = create_engine(
    "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server")

with engine.connect() as connection:
    with connection.begin():
        connection.execute("SELECT * FROM Customer")
```

Major Components - Transaction

Common methods:

- close() – Cancels the transaction, which performs a rollback.
- commit() – Commits the transaction.
- rollback() – Rolls back the transaction.

Major Components - ResultProxy

Façade around the DBAPI cursor object.

Returns rows via the RowProxy class.

```
engine = create_engine(
    "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server")

with engine.connect() as connection:
    with connection.begin():
        resultProxy = connection.execute("SELECT * FROM Customer")
```

Major Components - ResultProxy

Common methods:

- close() – Closes the underlying DBAPI cursor.
- fetchall() – Gets all rows from the cursor, and then closes the cursor.
- keys() – Returns a list of string keys represented by each row returned.

Major Components - ResultProxy

Common attributes:

- inserted_primary_key – Returns the primary key for the row just inserted. Only works on single row inserts.
- return_rows – True if the ResultProxy contains returned rows.
- rowcount – Number of rows that matched the WHERE condition of an UPDATE or DELETE statement.

Major Components - RowProxy

Represents a single result row.

Mostly acts like a Python named tuple, but also provides some Python dictionary behaviors.

```
engine = create_engine(
    "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server")

with engine.connect() as connection:
    with connection.begin():
        resultProxy = connection.execute("SELECT * FROM Customer")

        for row in resultProxy:
            print(row)
```

Major Components - RowProxy

Common methods:

- items() – Returns list of tuples of key (Column Name)/value (Column Value) pairs.
- keys() – Returns all the column names.
- values() – Returns all the column values.

Connecting to a Database

```
from sqlalchemy import create_engine

if __name__ == '__main__':
    engine = create_engine(
        "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server")
    conn = engine.connect()

    print(engine.table_names())
```

Result:

```
C:\Users\timgr\AppData\Local\Programs\Python\Python38-32\python.exe "D:/Projects/Python/PythonSqlAlchemyLunchAndLearn/Example 01 - Connect
['BuildVersion', 'ErrorLog']

Process finished with exit code 0
```

Changing the Schema

```
from sqlalchemy import create_engine

if __name__ == '__main__':
    engine = create_engine(
        "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server")
    conn = engine.connect()

    print(engine.table_names(schema="SalesLT"))
```

Result:

```
C:\Users\timgr\AppData\Local\Programs\Python\Python38-32\python.exe "D:/Projects/Python/PythonSqlAlchemyLunchAn
['Address', 'Customer', 'CustomerAddress', 'Product', 'ProductCategory', 'ProductDescription', 'ProductModel',

Process finished with exit code 0
```

Selecting Data

```python
from sqlalchemy import create_engine

if __name__ == '__main__':
    engine = create_engine(
        "mssql+pyodbc://python:python@localhost/AdventureWorksLT2019?driver=ODBC+Driver+17+for+SQL+Server")

    with engine.connect() as connection:
        result = connection.execute("select companyname from saleslt.customer")

        for row in result:
            print("Company Name: ", row['companyname'])
```

Result:

```
Company Name:  Mercantile Outlet
Company Name:  Exhilarating Cycles
Company Name:  Exercise Center
Company Name:  Great Bicycle Supply
Company Name:  Mail-Order Outlet
Company Name:  Nearby Cycle Shop
Company Name:  Retreat Inn
Company Name:  Technical Parts Manufacturing
Company Name:  Totes & Baskets Company
Company Name:  World of Bikes

Process finished with exit code 0
```

Example One

MetaData

Part of the SQLAlchemy Expression Language

Collection of Tables definitions and their Column definitions

Can be used to define tables and then create them

MetaData

bind() – Bind the MetaData to an Engine or Connection

clear() – Clears all the Tables from the MetaData only

create_all() – Creates all the tables defined in the bound database

drop_all() – Drops all the tables defined in the bound database

reflect() – Loads all table definitions from the bound database

MetaData

```
User = Table('user', metadata,
  Column('user_id', Integer, primary_key=True),
  Column('user_name', Integer, nullable=False),
  Column('email_address', String(60), key='email'),
  Column('nickname', String(50), nullable=False)
User_prefs = Table('user_prefs', metadata,
  Column('pref_id', Integer, primary_key=True),
  Column('user_id', Integer, ForeignKey("user.user_id"), nullable=False),
  Column('pref_name', String(40), nullable=False),
  Column('pref_value', String(100))
```

MetaData

Uses:

- Define and create tables
- Let SQLAlchemy define the SQL statements
- Selects, Inserts, Updates, Deletes
- Joins, Order By, Group By
- Where clause (AND, OR, NOT, LIKE)
- Aliases, CTEs, Unions

MetaData

```
>>> s = select([(users.c.fullname +
...                ", " + addresses.c.email_address).
...                label('title')]).\
...        where(
...           and_(
...               users.c.id == addresses.c.user_id,
...               users.c.name.between('m', 'z'),
...               or_(
...                  addresses.c.email_address.like('%@aol.com'),
...                  addresses.c.email_address.like('%@msn.com')
...               )
...           )
...        )
>>> conn.execute(s).fetchall()
```

```
SELECT users.fullname || ? || addresses.email_address AS title
FROM users, addresses
WHERE users.id = addresses.user_id AND users.name BETWEEN ? AND ? AND
(addresses.email_address LIKE ? OR addresses.email_address LIKE ?)
(', ', 'm', 'z', '%@aol.com', '%@msn.com')
```

```
[(u'Wendy Williams, wendy@aol.com',)]
```

Example Two

ORM - Session

All conversations to the database go through this

Establishes Transactions with each new Session object

Keeps track of all loaded objects and changes to them, i.e. new, dirty

Uses Unit of Work pattern internally

ORM - Session

Common methods:

- add() – Adds an object to the session
- add_all() – Adds multiple objects in one call
- begin() – Begin a new transaction
- close() – Close the session
- commit() – Commits the current transaction
- delete() – Deletes and object

ORM - Session

Common methods continued:

- flush() – Flushes all changes to the database
- query() – Returns a new Query object
- rollback() – Rolls back the current transaction
- close() – Close the session
- commit() – Commits the current transaction
- delete() – Deletes and object

ORM - Base

Base class for all database table objects declared

Includes all the necessary SQLAlchemy tools for keeping track of new, dirty, delete, etc.

Known as the Declarative Base class using SQLAlchemy’s Declarative system

Base class is declared once, and imported from a common module.

ORM - Base

Minimum necessary is

- __tablename__ attribute
- One Column that is part of a primary key

ORM – Class/Table Definition

```
>>> from sqlalchemy import Column, Integer, String
>>> class User(Base):
...     __tablename__ = 'users'
...
...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     fullname = Column(String)
...     nickname = Column(String)
...
...     def __repr__(self):
...        return "<User(name='%s', fullname='%s', nickname='%s')>" % (
...                             self.name, self.fullname, self.nickname)
```

ORM – Querying Objects

```
>>> for instance in session.query(User).order_by(User.id):
...     print(instance.name, instance.fullname)
ed Ed Jones
wendy Wendy Williams
mary Mary Contrary
fred Fred Flintstone
```

SQL

```
>>> for u in session.query(User).order_by(User.id)[1:3]:
...     print(u)
<User(name='wendy', fullname='Wendy Williams', nickname='windy')>
<User(name='mary', fullname='Mary Contrary', nickname='mary')>
```

SQL

```
>>> for user in session.query(User).\
...             filter(User.name=='ed').\
...             filter(User.fullname=='Ed Jones'):
...     print(user)
<User(name='ed', fullname='Ed Jones', nickname='eddie')>
```

SQL

ORM – Adding/Updating Objects

```
>>> ed_user = User(name='ed', fullname='Ed Jones', nickname='edsnickname')
>>> session.add(ed_user)
```

```
>>> ed_user.nickname = 'eddie'
```

```
>>> session.add_all([
...     User(name='wendy', fullname='Wendy Williams', nickname='windy'),
...     User(name='mary', fullname='Mary Contrary', nickname='mary'),
...     User(name='fred', fullname='Fred Flintstone', nickname='freddy')])
```

Example Three

ORM – Personal Failure

- Best part of ORMs is the relationships
- Query for Sales Order Header and get Sales Order Detail as a child object in the parent object
- Capability Exists
- Couldn’t get it working with existing database

ORM – Simple Example

- Made a new database with 3 tables
  - Publisher – ID, Name, Website
  - Author – ID, First Name, Last Name, Rating
  - Book – ID, Publisher ID, Author ID, Name, Rating\

- Only Foreign Keys is on Book
  - Publisher ID
  - Author ID

ORM – The Key

- The key was circular relationships!
- Book is setup as related to Publisher and Author
- Publisher and Author have no relationship to Book until runtime, when the relationship is added
- You can query any of the three objects, and pull in the other objects with one query statement
- Finally, a usable ORM implementation

Example Four

Code

Github Repository:

https://github.com/tgroven/PythonSqlAlchemyLunchAndLearn