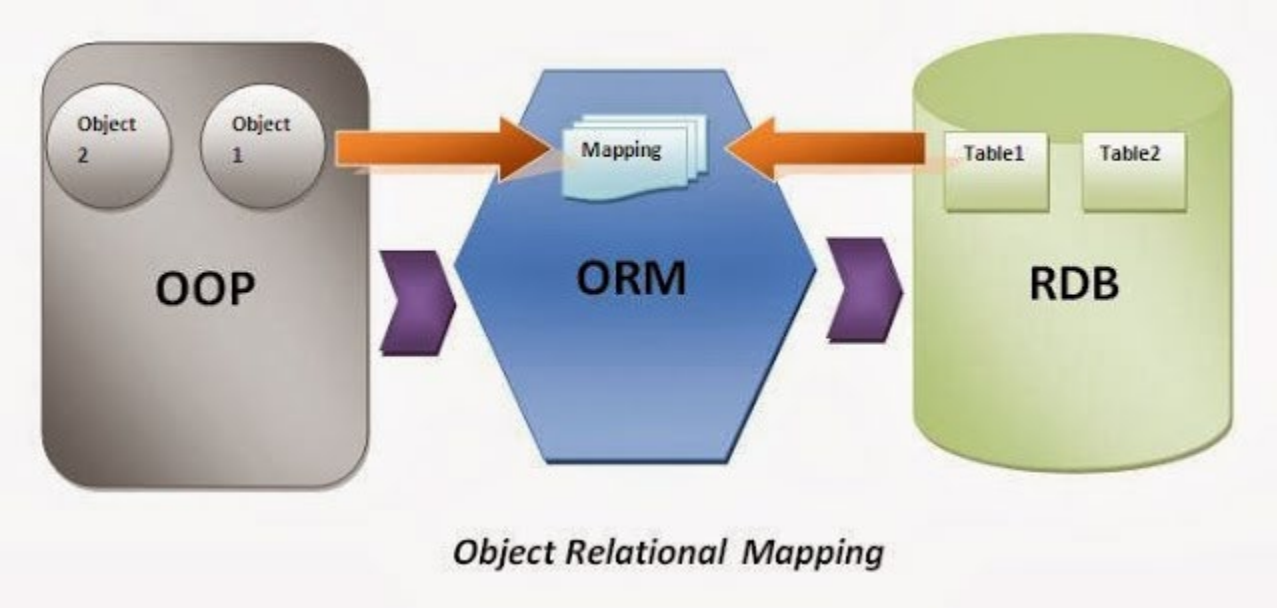
SQLAlchemy ORM with Flask

Cyrus Wong

Source:
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-ii-templates
HarvardX CS50x - CS50's Introduction to Computer Science

Outline

- Object-Oriented Programming (OOP)
- Object-Relational Mapping (ORM)
- SQL vs SQLAlchemy
- ORM Relationship



Prerequisites

- Knowledge & experience in Python 3.6 or higher and Flask

Object-Oriented Programming

- Python, along with many other programming languages, use Object-Oriented Programming (OOP).
- An ‘object’ is a discrete item.
- OOP allows for the creation of classes, which are the generic forms of objects.

Object-Oriented Programming

- For example, a ‘flight’ class is defines:
  - All the components which describe a flight, - - Actions to take (such as adding a passenger)
- Similarly, a ‘passenger’ class would represent the generic idea of passenger, defines:
  - Passenger name
  - Associated flight number

Object-Oriented Programming

- Here's a simple example of a Python class.

```
class Flight:
    def __init__(self, origin, destination, duration):
        self.origin = origin
        self.destination = destination
        self.duration = duration
```

Constructor

- __init__ is a **constructor** 'method', which is a function performed on individual objects.
- __init__ in particular is a special, built-in method that describes what should happen when a flight object is created.
- Taking **self** as their first argument. **self** refers to the object being worked with.
- The other three arguments are simply the information that should be stored about a particular flight.
- That information is stored as 'properties' inside the object, using dot notation.

Objects

- Here's how the Flight class might be used:

```
# Create flight.
f = Flight(origin="New York", destination="Paris", duration=540)
# Change the value of a property.
f.duration += 10    # Print details about flight.
print(f.origin)
print(f.destination)
print(f.duration)
```

A flight object "f" is being created with Fight class definition

Methods

- Methods are functions define within the class
- Additional methods can be added besides constructor method __init__

```
class Flight:
    # assume same __init__ method
    ...
    def print_info(self):
        print(f"Flight origin: {self.origin}")
        print(f"Flight destination: {self.destination}")
        print(f"Flight duration: {self.duration}")

    def main():
        f1 = Flight(origin="New York", destination="Paris", duration=540)
        f1.print_info()
```

self refers to the object that the method is being called on. In this example, that's f1.

Printing out object flight info

Methods

- Methods can also take additional arguments and modify properties.

```
def delay(self, amount):
    self.duration += amount
```

* Note that writing methods need to be logical and easily understood way, without needing to know or even understand how Class being implemented.

E.g Flight <-> delay, Flight <-> print_info

Another Example

- Given a simple Passenger class…

```
class Passenger:
    def __init__(self, name):
        self.name = name
```

Advanced Example

- A more complex Flight class can be implemented.

  A walkthrough of “advanced_class.py”

Advanced Example

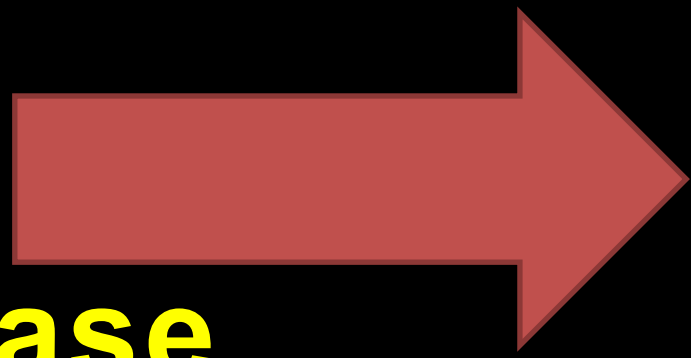
- Here’s how the more advanced Flight class could be used:

```
# Create flight.
   flight = Flight(origin='New York', destination='Paris', duration=540)

   # Create passengers.
   alice = Passenger(name='Alice')
   bob = Passenger(name='Bob')

   # Add passengers.
   flight.add_passenger(alice)
   flight.add_passenger(bob)
   flight.add_passenger(Passenger(name='Charlie'))

   # Print flight information.
   flight.print_info()
```

Object-Relational Mapping (ORM)

- Allows for the combination of:
  - OOP world of **Python**
  - Relational database world of **SQL**.
- ORM = OOP + SQL
- With ORM, Python classes, methods, and objects become the tools for interacting with SQL databases.
- To do this, the Flask-SQLAlchemy package will be used.

Object-Oriented Programming



Relational Database Mapping

Python
SQL
Mapping

SQL

~~SQL~~

Python

Flask-SQLAlchemy

- Extension for Flask that adds support for SQLAlchemy
- SQLAlchemy is the Python SQL toolkit and Object Relational Mapper
- 1 Python Class - 1 Database Table
- SQLAlchemy 3.x <-> Flask-SQLAlchemy 2.x

Create Table in SQL

```
CREATE TABLE flights   (
    id SERIAL PRIMARY KEY,
    origin VARCHAR NOT NULL,
    destination VARCHAR NOT NULL,
    duration INTEGER NOT NULL
);
```

Flask-SQLAlchemy 1.X

- The basic setup for SQLAlchemy 2.x

```
from flask_sqlalchemy import SQLAlchemy
db = SQLAlchemy()

class Flight(db.Model):
    __tablename__ = "flights”
    id = db.Column(db.Integer, primary_key=True)
    origin = db.Column(db.String, nullable=False)
    destination = db.Column(db.String, nullable=False)
    duration = db.Column(db.Integer, nullable=False)
class Passenger(db.Model):
    __tablename__ = "passengers"
   id = db.Column(db.Integer, primary_key=True)
   name = db.Column(db.String, nullable=False)
   flight_id = db.Column(db.Integer, db.ForeignKey("flights.id"), nullable=False)
```

Flask-SQLAlchemy 2.X

- The basic setup for SQLAlchemy 3.x

```
from sqlalchemy import String
from sqlalchemy.orm import DeclarativeBase
from sqlalchemy.orm import Mapped, mapped_column
class Base(DeclarativeBase):
    pass
class Flight(Base):
    __tablename__ = "flight”
    id: Mapped[int] = mapped_column(primary_key=True)
    origin: Mapped[str] = mapped_column(String(30))
    destination: Mapped[str] = mapped_column(String(30))
    duration : Mapped[int] = mapped_column()
class Passenger(Base):
    __tablename__ = "passenger"
   id: Mapped[int] = mapped_column(primary_key=True)
   name: Mapped[str] = mapped_column(String(150))
   flight_id: Mapped[int] = mapped_column(ForeignKey("flight.id"))
```

Flask-SQLAlchemy

- **(Old) db.model** in parentheses after class names indicates that these classes ‘inherit’ from db.Model.
  * inheritance are unimportant right now; simply, this allows for the class to have some built-in relationship with SQLAlchemy to interact with the database

```
class Flight(db.Model):
```

- **__tablename__** naturally corresponds with the table name inside the database.

```
__tablename__ = "flights”
```

Flask-SQLAlchemy

- Every property is defined as a db.Column, which will become columns in the table. The arguments

```
flight_id = db.Column(db.Integer, db.ForeignKey("flights.id"), nullable=False) #Old
```

```
flight_id: Mapped[int] = mapped_column(ForeignKey("flights.id"))       #New
```

- Note that flights.id is marked as a foreign key using the __tablename__ flights, not the class name Flight.

Flask-SQLAlchemy (New Verison)

```
from sqlalchemy import Integer, String, ForeignKey
from sqlalchemy.orm import DeclarativeBase
from sqlalchemy.orm import Mapped
from sqlalchemy.orm import mapped_column

# declarative base class
class Base(DeclarativeBase):
    pass

# an example mapping using the base
class User(Base):
    __tablename__ = "user"

    id: Mapped[int] = mapped_column(primary_key=True)
    name: Mapped[str]
    fullname: Mapped[str] = mapped_column(String(30))
    nickname: Mapped[Optional[str]]
```

Nullable

For more detail about Model, please check out
Object-oriented Programming
for Database in Python Lecture

SQL vs SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort

```
CREATE TABLE flights (
        id SERIAL PRIMARY KEY,
        origin VARCHAR NOT NULL,
        destination VARCHAR NOT NULL,
        duration INTEGER NOT NULL
);
```

```
class Flight(db.model):

  tablename = "flights"

  id = db.Column(db.Integer, primary_key=True)

  origin = db.Column(db.String, nullable=False)

  destination = db.Column(db.String, nullable=False)

  duration = db.Column(db.Integer, nullable=False)
```

```
from sqlalchemy import Integer, String
from sqlalchemy.orm import Mapped, mapped_column

class Flight(db.Model):
    __tablename__ = "flight”
    id: Mapped[int] = mapped_column(primary_key=True)
    origin: Mapped[str] = mapped_column(String(30))
    destination: Mapped[str] = mapped_column(String(30))
    duration : Mapped[int] = mapped_column()
```

```
CREATE TABLE flights (
    id SERIAL PRIMARY KEY,
    origin VARCHAR NOT NULL,
    destination VARCHAR NOT NULL,
    duration INTEGER NOT NULL
);
```

```
db.create_all() #Old
Base.metadata.create_all(engine) #New
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort

```
INSERT INTO flights
(origin, destination, duration)
VALUES
('New York', 'Paris', 540);
```

```
flight =  Flight(origin = 'New York',
                 destination = 'Paris',
                 duration = 540)
db.session.add(flight)
```

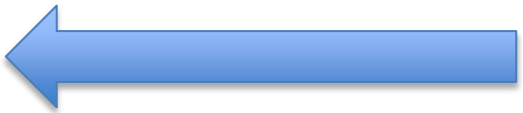
SQL vs SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort

```
SELECT * FROM flights;
```

```
Flight.query.all()
```

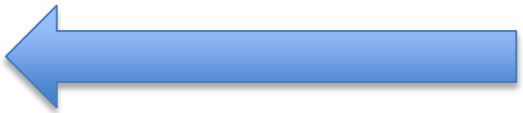
SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter
- Update
- Delete
- Commit
- Sort

```
SELECT * FROM flights
WHERE origin = 'Paris';
```

```
Flight.query.filter_by(origin='Paris').all()
```

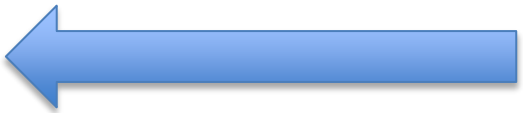
SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter, Limit
- Update
- Delete
- Commit
- Sort

```
SELECT * FROM flights
WHERE origin = 'Paris'
LIMIT 1;
```

```
Flight.query.filter_by(origin='Paris').first()
```

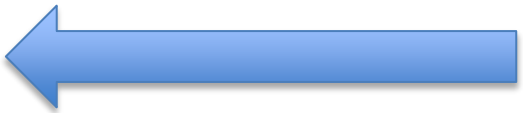
SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter, Count
- Update
- Delete
- Commit
- Sort

```
SELECT COUNT(*) FROM flights
WHERE origin = 'Paris';
```

```
Flight.query.filter_by(origin='Paris').count()
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter only 1
- Update
- Delete
- Commit
- Sort

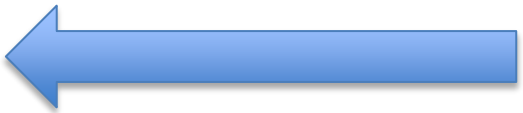
```
SELECT * FROM flights
WHERE id = 44;
```

```
Flight.query.filter_by(id=44).first()
```

```
SELECT * FROM flights
WHERE id = 44;
```

```
Flight.query.get(44)
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter, Not Equal to (!=)
- Update
- Delete
- Commit
- Sort

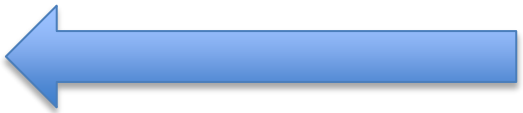
```
SELECT * FROM flights
WHERE origin != 'Paris';
```

```
Flight.query.filter(
    Flight.origin != 'Paris').all()
```

```
SELECT * FROM flights
WHERE origin != 'Paris';
```

```
Flight.query.filter(
    Flight.origin != 'Paris').all()
```

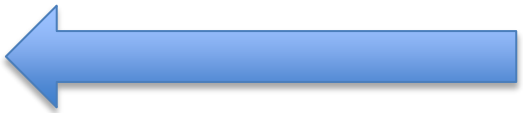
SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter, LIKE
- Update
- Delete
- Commit
- Sort

```
SELECT * FROM flights
WHERE origin LIKE '%a%';
```

```
Flight.query.filter(
    Flight.origin.like('%a%')).all()
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter, IN
- Update
- Delete
- Commit
- Sort

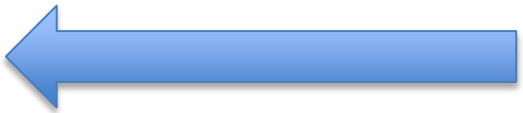
```
SELECT * FROM flights
WHERE origin
IN ('Tokyo', 'Paris');
```

```
Flight.query.filter(
    Flight.origin.in_(
        ['Tokyo', 'Paris'])).all()
```

```
SELECT * FROM flights
WHERE origin
IN ('Tokyo', 'Paris');
```

```
Flight.query.filter(
    Flight.origin.in_(
        ['Tokyo', 'Paris'])).all()
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter, AND
- Update
- Delete
- Commit
- Sort

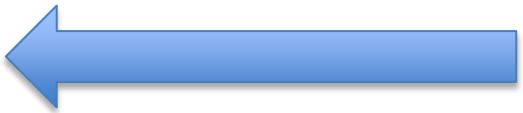
```
SELECT * FROM flights
WHERE origin = 'Paris' AND
duration > 500;
```

```
Flight.query.filter(
    and_(Flight.origin == 'Paris',
         Flight.duration > 500)).all()
```

```
SELECT * FROM flights
WHERE origin = 'Paris' AND
duration > 500;
```

```
Flight.query.filter(
    and_(Flight.origin == 'Paris',
         Flight.duration > 500)).all()
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query  Filter, OR
- Update
- Delete
- Commit
- Sort

```
SELECT * FROM flights
WHERE origin = 'Paris' OR
duration > 500;
```

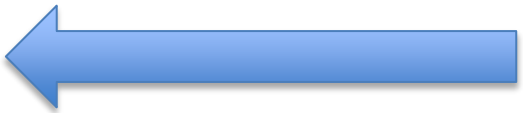
```
Flight.query.filter(
    or_(Flight.origin == 'Paris',
        Flight.duration > 500)).all()
```

```
SELECT * FROM flights
WHERE origin = 'Paris' OR
duration > 500;
```

```
Flight.query.filter(
    or_(Flight.origin == 'Paris',
        Flight.duration > 500)).all()
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort



Filter,
JOIN (more than 1 tables)

```
SELECT * FROM flights JOIN passengers
ON flights.id = passengers.flight_id;
```

```
db.session.query(Flight, Passenger).filter(
    Flight.id == Passenger.flight_id).all()
```

ORM Relationships

- One more powerful feature of ORMs is the idea of relationships.
- SQLAlchemy relationships are an easy way to take one table and relate it to another table, such that the each can refer to the other.
- A relationship is set up with a single line, which in this case would be added to the definition of the Class.

ORM Relationships

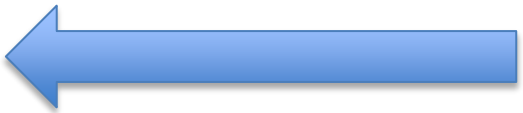
- passengers = db.relationship("Passenger", backref="flight", lazy=True)
  - passengers is not a column, but rather just a relationship. Given a flight object, the passengers property can be used to extract all the passenger info for that flight.
  - backref creates a relationship in the opposite direction, from Flight to Passenger.
  - lazy indicates that the information should be fetched only when it's asked for.

ORM Relationships

- With those relationships set up, the code in application.py's flight function to list get all passengers is extremely simplified.
- passengers = flight.passengers

SQL vs SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort



Filter,
JOIN (more than 1 tables)

This time using the concept of ORM Relationships!

```
SELECT * FROM passengers
WHERE flight_id = 1;
```

```
Flight.query.get(1).passengers
```

```
SELECT * FROM flights JOIN passengers
ON flights.id = passengers.flight_id
WHERE passengers.name = 'Alice';
```

```
Passenger.query.filter_by(name='Alice').
first().flight
```

```
SELECT * FROM flights JOIN passengers
ON flights.id = passengers.flight_id
WHERE passengers.name = 'Alice';
```

```
Passenger.query.filter_by(name='Alice').
first().flight
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort

```
UPDATE flights
SET duration = 280
WHERE id = 6;
```

```
flight = Flight.query.get(6)
flight.duration = 280
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort

```
DELETE FROM flights
WHERE id = 44;
```

```
flight = Flight.query.get(44)
db.session.delete(flight)
```

SQL vs Flask-SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort

COMMIT

```
db.session.commit()
```

SQL vs SQLAlchemy

- Create Table
- Insert
- Query
- Update
- Delete
- Commit
- Sort

```
SELECT * FROM flights
ORDER BY origin;
```

```
Flight.query.order_by(Flight.origin).all()
```

```
SELECT * FROM flights
ORDER BY origin DESC;
```

```
Flight.query.order_by(Flight.origin.desc()).all()
```

Model View Controller Pattern