SQLAlchemy

What is an ORM?

- Until now we have been using pure SQL queries to interact with the database
- ORM, or Object Relational Mapping, is a layer of abstraction on top of relational database and its query language
- Instead of operating directly on databases, queries and rows, we use objects, filters and iterators to represent the underlying data.

What is SQLAlchemy?

- SQLAlchemy is one of the most popular Python ORMs
- It uses purely Pythonic objects to represent the data:
  - class represents a table
  - instance represents a row
  - instance attribute represents a column
- Session is the interface for communicating with the database in SQLAlchemy
- `pip install sqlalchemy`

Creating a table

To create a table you have to:

- Connect to the database using `create_engine()`
- Create a declarative base class
- Use it as a superclass for classes defining tables
- Use `Base.metadata.create_all(engine)` to create all our tables

Example 02-sqlalchemy/example-01.py

```
from sqlalchemy import create_engine
from models import Base, Student

CONNECTION_STRING = "mysql+pymysql://{user}:{password}@{host}/{db}"

eng = create_engine(
    CONNECTION_STRING.format(
        user="root", password="password", host="127.0.0.1", db="default"
    )
)
Base.metadata.create_all(eng)
```

```
from sqlalchemy.ext.declarativeimport declarative_base
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime


Base = declarative_base()


class Student(Base):
    __tablename__ = "students"
    id = Column(Integer, primary_key=True, autoincrement=True)
    first_name = Column(String(255))
    last_name = Column(String(255))

    def __str__(self):
        return f"<Student #{self.id} {self.first_name} {self.last_name}>"
```

```
$ python3 02-sqlalchemy/example-01.py
```

Adding data

- As mentioned before, you need a session object
- To do so, use `sessionmaker` function to create Session base class
- Then create a `Session` instance
- Use the class we just created to add some new students with `add_all(list)` or `add(object)`

Example 02-sqlalchemy/example-02.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import Base, Student

CONNECTION_STRING = "mysql+pymysql://{user}:{password}@{host}/{db}"

eng = create_engine(
    CONNECTION_STRING.format(
        user="root", password="password", host="127.0.0.1", db="default"
    )
)
Session = sessionmaker(bind=eng)
s = Session()

s.add_all(
    [
        Student(first_name="Mike", last_name="Wazowski"),
        Student(first_name="Netti", last_name="Nashe"),
        Student(first_name="Jessamine", last_name="Addison"),
        Student(first_name="Brena", last_name="Bugdale"),
        Student(first_name="Theobald", last_name="Oram"),
    ]
)
s.commit()
```

```
$ python3 02-sqlalchemy/example-02.py
```

Querying data

- To query the data, use session object like so: `session.query(<class>).all()`
- You can use existing classes which inherit after the Base class
- You can use a `filter(<class>.property == <value>)` function to create SQL like clauses (similar to `WHERE`)

Example 02-sqlalchemy/example-03.py

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import Base, Student
CONNECTION_STRING = "mysql+pymysql://{user}:{password}@{host}/{db}"
eng = create_engine(
    CONNECTION_STRING.format(
        user="root", password="password", host="127.0.0.1", db="default"
    )
)
Session = sessionmaker(bind=eng)
s = Session()

rows = s.query(Student).all()
for row in rows:
    print(row)

print("---")
total = s.query(Student).count()
print(f"Total: {total}")

print("---")
query_result = s.query(Student).filter(Student.id >= 2,
Student.first_name.like("Bre%"))
print("Found students:")
for row in query_result:
    print(row)
```

```
$ python3 02-sqlalchemy/example-03.py
```

```
<Student #1 Mike Wazowski>
<Student #2 Netti Nashe>
<Student #3 Jessamine Addison>
<Student #4 Brena Bugdale>
<Student #5 Theobald Oram>
---
Total: 5
---
Found students:
<Student #4 Brena Bugdale>
```

Foreign keys

- to define foreign keys, add `ForeignKey(<class.attribute>)` to the `Column` declaration
- to query with joins on tables using foreign keys, use `session.query(class1).join(class2)`
- to get data from both tables, use `session.query(class1, class2).join(class2)`

Example 02-sqlalchemy/example-04.py

```
from sqlalchemy.exc import IntegrityError, InvalidRequestError
(...)
try:
    s.add_all([
            Locker(number=1, student=4),
            Locker(number=2, student=1),
            Locker(number=3, student=5),
            Locker(number=4, student=2),
            Locker(number=5, student=3),
    ])
    s.commit()
except IntegrityError:
    s.rollback()
    print("Lockers already created!")

rows = s.query(Student, Locker).join(Locker).filter(Locker.number == 4)
for row in rows:
    student, locker = row
    print(f"Student with locker #{locker.number}: {student}")
```

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime


Base = declarative_base()


class Locker(Base):
    __tablename__ = "lockers"
    number = Column(Integer, primary_key=True)
    student = Column(Integer, ForeignKey(Student.id), primary_key=True)

    def __str__(self):
        return f"<Locker {self.number}: {self.student}>"
```

```
$ python3 02-sqlalchemy/example-04.py
```

```
Student with locker #4: <Student #2 Netti Nashe>
```

Exercise 1

- Create a new table called `address`
- It should include the following fields:
  - student **int, foreign key, primary key**
  - street_name **string**
  - number **int**
  - city **string**
- Add an `Address` for each student
- Print out all students along with their addresses using a `join()`

Exercise 2

- Create a table called `grades`
- It should include the following fields:
  - id **int, primary key, autoincrement**
  - student **int, foreign key**
  - grade **int** or **string -** whichever you prefer
  - date_created **datetime**
- Add some grades for each student
- Print out all grades per each student using `filter`