COMP 3005 Final Project Report
Cal McLelland
100858499

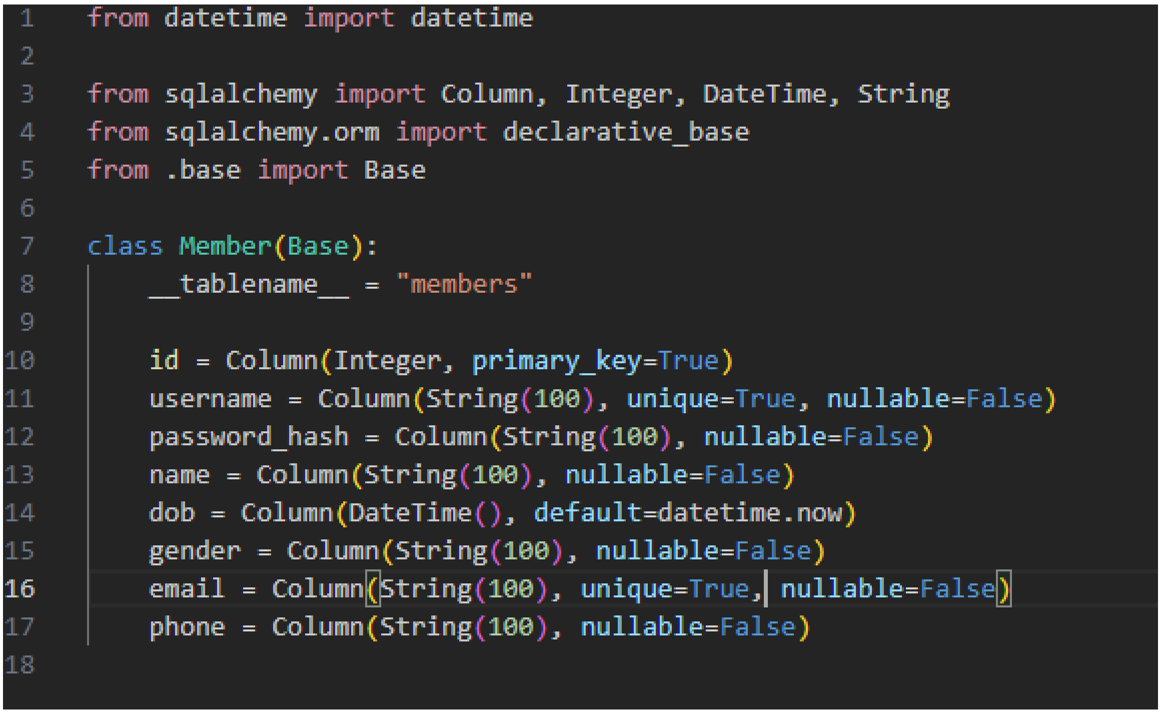
ORM Usage

My project has been implemented using Python and SQLAlchemy, allowing me to use ORM for interactions with the database.

The entities mapped using ORM are listed here:

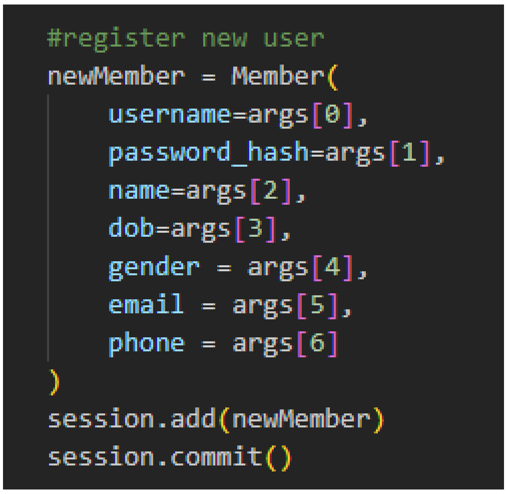
- Member
- Trainer
- Admin
- Health_Metric
- Availability
- Class
- Session
- Room
- Takes

For data definition, the object files exist in the models folder. Here is one example, member.py:



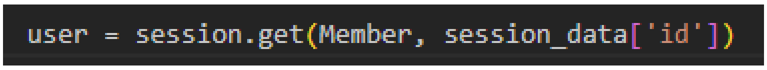
```python
from datetime import datetime

from sqlalchemy import Column, Integer, DateTime, String
from sqlalchemy.orm import declarative_base
from .base import Base

class Member(Base):
    __tablename__ = "members"

    id = Column(Integer, primary_key=True)
    username = Column(String(100), unique=True, nullable=False)
    password_hash = Column(String(100), nullable=False)
    name = Column(String(100), nullable=False)
    dob = Column(DateTime(), default=datetime.now)
    gender = Column(String(100), nullable=False)
    email = Column(String(100), unique=True, nullable=False)
    phone = Column(String(100), nullable=False)

```

For adding new entries in the database, objects are created and then added, as shown here for a member object:



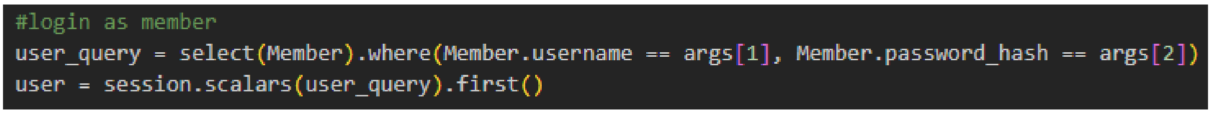
```
#register new user
newMember = Member(
    username=args[0],
    password_hash=args[1],
    name=args[2],
    dob=args[3],
    gender = args[4],
    email = args[5],
    phone = args[6]
)
session.add(newMember)
session.commit()
```

For data querying, data can be accessed directly if the ID of the object is known, as demonstrated here:



```
user = session.get(Member, session_data['id'])
```

If the ID of the object is not known, a query must be used; however, SQLAlchemy provides wrappers so that raw SQL does not need to be used. An example is shown below, which comes from the login process. In this case, the username and password are used for the query:



```
#login as member
user_query = select(Member).where(Member.username == args[1], Member.password_hash == args[2])
user = session.scalars(user_query).first()
```

Normalization

1st Normal Form

Since every attribute in each relation is atomic, the database is in 1st normal form.

2nd and 3rd Normal Form

For 2nd and 3rd Normal Form, we must analyze the functional dependencies:

for the **Member, Trainer,** and **Administrator** relations:

{id} -> username
{id} -> password_hash
{id} -> name
{id} -> dob
{id} -> gender
{id} -> email
{id} -> phone

contains no partial dependencies or transitive dependencies, and therefore satisfies 2nd and 3rd Normal Form.

for the **Health_Metric** relation:

{date, record_type, user_id} -> weight
{date, record_type, user_id} -> height
{date, record_type, user_id} -> vo2Max
{date, record_type, user_id} -> body_composition
{date, record_type, user_id} -> resting_hr

contains no partial dependencies or transitive dependencies, and therefore satisfies 2nd and 3rd Normal Form.

for the **Class** relation:

{id, room_id, start_time} -> end_time
{id, room_id, start_time} -> trainer_id

contains no partial dependencies or transitive dependencies, and therefore satisfies 2nd and 3rd Normal Form.

for the **Session** relation:

{id, room_id, start_time} -> end_time
{id, room_id, start_time} -> trainer_id
{id, room_id, start_time} -> member_id

contains no partial dependencies or transitive dependencies, and therefore satisfies 2nd and 3rd Normal Form.

for the **Room** relation:

{id} -> room_number

contains no partial dependencies or transitive dependencies, and therefore satisfies 2nd and 3rd Normal Form.

for the **Takes** relation:

{member_id, class_id}

contains no partial dependencies or transitive dependencies, and therefore satisfies 2nd and 3rd Normal Form.