ADInt 2021/2022

Laboratory 4

In this laboratory students will learn how to use an ORM no access a database.

The provided code illustrates how to use SQLAlchemy to store information about authors and corresponding books.

1 SQLAlchemy

https://docs.sqlalchemy.org/

https://www.tutorialspoint.com/sqlalchemy/index.htm

SQLAlchemy is a popular SQL toolkit and Object Relational Mapper. It is written in Python and gives full power and flexibility of SQL to an application developer.

ORM (Object Relational Mapping) is a programming layer that converts data between incompatible type systems in object-oriented programming languages, usually it is used to convert the programming language data-types and access modes (obj.attribute) to SQL data-types and queries.

In an ORM system, each class maps to a table in the underlying database. The programmer, instead of writing tedious database queries code himself, uses the ORM that takes care of these issues.

To use the sqlAlchemy it is necessary to install its package

```
pip install sqlalchemy
```

SQLAlchemy is designed to operate with drivers built specific databases. Drivers for the following databases are available:

- Firebird
- Microsoft SQL Server
- MySQL
- Oracle
- PostgreSQL
- SQLite
- Sybase

1.1 Declare Mapping

https://www.tutorialspoint.com/sqlalchemy/sqlalchemy_orm_declaring_mapping.htm

https://docs.sqlalchemy.org/en/13/orm/tutorial.html#declare-a-mapping

In order to use SQLAlchemy it is firs necessary to define the data mappings.

The programmer defines a set of classes that will be:

- used in the program to access data
- used by SQLAlchemy to translate the python objects to SQL.

```
class Author(Base):
  __tablename__ = 'author'
  id = Column(Integer, primary_key=True)
  name = Column(String)
  dateBirth = Column(Date) # A type for datetime.date() objects.
  def __repr__(self):
    return "Author(id=%d name='%s' dateBirth='%s' n_books=%d)" %
      (self.id, self.name, str(self.dateBirth), len(self.books))
```

The previous example shows a class being defined.

The __**tablename**__ defined the name of the table on the database

Each attribute will be assigned one column (with the defined type) in the table.

The **id** attribute is a foreign key and will be initialized automatically.

In the definition of the Book, the programmer defined the key as the ISBN, a string that is supplied at the creation of the object.

```
class Book(Base):
    __tablename__ = 'books'
    isbn = Column(String, primary_key=True)
    title = Column(String, nullable=False)
    publisher = Column(String, nullable=False)
    author_id = Column(Integer, ForeignKey('author.id'))
```

1.2 Table relationships

https://docs.sqlalchemy.org/en/13/orm/tutorial.html#building-a-relationship

The next example shows a creation of a relation between two tables (Book and Author):

```
class Book(Base):
    __tablename__ = 'books'
```

```
    isbn = Column(String, primary_key=True)
    title = Column(String, nullable=False)
    publisher = Column(String, nullable=False)
    author_id = Column(Integer, ForeignKey('author.id'))
    author = relationship("Author", back_populates="books")
Author.books = relationship("Book", ... , back_populates="author")
```

In this example each book will have one author.

This relation is defined by the line:

```
author_id = Column(Integer, ForeignKey('author.id'))
```

The **Foreignkey** statement tells SQLAlchemy what is the identifier of author (in this case **id**) that will be stored in the book record.

To further ease the access of this related objects the programmer should also insert the following statement:

```
author = relationship("Author", back_populates="books")
```

inside the Book class, and

```
Author.books =relationship("Book",order_by=Book.isbn, back_populates="author")
```

after the Book class.

These two statements allow SQLAlchemy to automatically update the Book and Author objects when the **author_id** is assigned.

1.3 Object creation

https://www.tutorialspoint.com/sqlalchemy/sqlalchemy_orm_adding_objects.htm

https://docs.sqlalchemy.org/en/13/orm/tutorial.html#adding-and-updating-objects

To create records in the database, first the programmer creates an python object from the previous classes and then tell SQLAlchemy to store it in the databases.

```
auth1 = Author(name="Camões", dateBirth=datetime.date(1524,1,1))
session.add(auth1)
session.commit()
```

The first line creates a python object that represents an Author.

The second line invokes the object transformation into SQL and updates the tables

the third line persists the changes in the database. If the commit is not executed, when restarting the application no object was put into the database

In order to create one relation between object it is only necessary to assign the correct attribute:

```
book1 = Book(isbn = "9789897021824", title = 'Os Lusíadas',
       publisher="Guerra & Paz", reserved = False, author_id = 1)
```

The previous example creates a book whose author has the id of 1.

SQL Alchemy will create a book and make a relation between this book and the author with id of 1.

It is then necessary to add the object to the database and execute the commit.

1.4 Querying

https://docs.sqlalchemy.org/en/13/orm/tutorial.html#querying

https://www.tutorialspoint.com/sqlalchemy/sqlalchemy_orm_using_query.htm

https://www.tutorialspoint.com/sqlalchemy/sqlalchemy_orm_applying_filter.htm

https://www.tutorialspoint.com/sqlalchemy/sqlalchemy_orm_filter_operators.htm

SQLAlchemy provides Query objects that access one particular table and allows the programmer to retrieve the objects that fill certain criteria.

This example

```
q1 = session.query(Author)
listAuthors = q1.all()
```

returns a list with all the Authors.

By changing the final method, it is possible to return different objects:

```
print( q1.first()) # the first record
print( q1.count() # the number of records
```

Tho filter the record that should be returned the programmer should use the filter method:

```
q2 = session.query(Author).
          filter(Author.dateBirth < datetime.date(1900, 1, 1))
oldAuthors = q2.all()
q3 = session.query(Author).filter(Author.id==1)
authx = q3.first()
```

it is even possible to use objects returned by a query in the filter:

```
q1 = session.query(Author)
camoes =q1.filter(Author.id==1).first()
booksCamoes=q1.filter(Book.author== camoes).all()
```

1.5 updating objects

https://www.tutorialspoint.com/sqlalchemy/sqlalchemy_orm_updating_objects.htm

In order to update a database record it is only necessary to retrieve the corresponding object, change it using regular python instructions and the call the commit function:

```
q1 = session.query(Author)
camoes =q1.filter(Author.id==1).first()
camoes.name =  "Luis Vaz de Camões"
session.commit()
```

2 Exercise

Modify the work done on the previous laboratory so that the application stores When each file was downloaded.

Students should use SQLAlchemy to manage a database with one table. This database table will store one record every time each file is downloaded.

To accomplish this, students should follow the next steps

2.1 Database initialization

When stating the server it is necessary to define what database system to be used (SQLite is good because all data is stored in files) and initialize the SQLAlchemy middleware.

2.2 Definition of the data model

Students should define the python classes (as described in 1.1) that will allow the storage of the information about each file download:

- file name
- date/time of download
- IP address of the client that downloaded the file

2.3 Insertion of download data

Each time a file is downloaded the python code should create a new object and insert it in the database

2.4 Show number of downloads

When listing the available files user should also see the number of downloads

2.5 Download Listing

Students should create a new set of pages (one for each file) that lists the date/time of all downloads of such file.

Students should implement a page that shows in HTML such information and another page that shows it in JSON.