Using an ORM - Lab

Introduction

In this lab, we'll make use of SQLAlchemy to execute CRUD operations on a SQL database!

Objectives

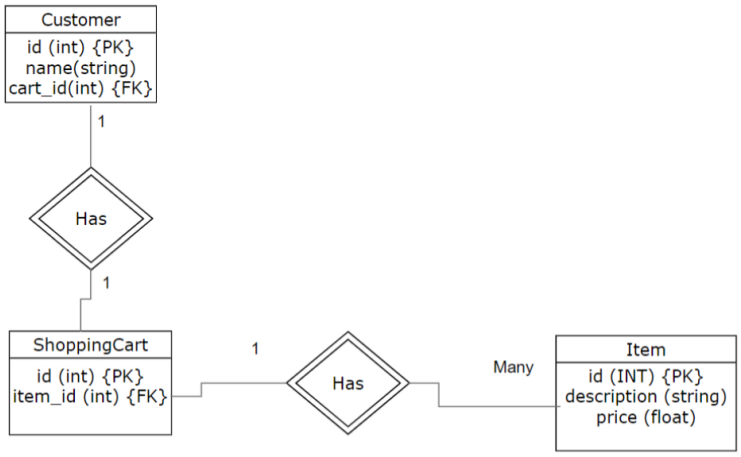
You will be able to:

- Identify the steps needed to use SQLAlchemy with a database
- Understand and explain the concept of an Object Relational Mapper
- Execute CRUD operations on a database using SQLAlchemy

Getting Started

In this lesson, we'll make use of our newfound SQLAlchemy knowledge to create a database, populate it with data, and write queries to retrieve objects containing the information we want.

We'll start by setting up our database. For this lesson, we're going to create then database described in the following ERD:



Question:

What sort of relationship do customers have with shopping carts? What sort of relationship do shopping carts have with items?

Write your answer below this line:

Customers have a 1-to-1 relationship with shopping carts, while shopping carts have 1-to-many relationship with items.

Defining Our Mappings

We'll begin by importing everything we need to create our database and structure our mappings so that they look like the tables in the ERD.

In the cell below:

- Import everything from sqlalchemy
- Import `declarative_base`
- Create a `Base` object

In [1]:
```
from sqlalchemy import *
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
```

Good! Now, since we'll need to define relationships between our tables, we'll need to import one more thing. In the cell below, import `relationship` from sqlalchemy's `orm` module.

Note: Make sure you import `relationship`, not the plural `relationships`!

In [2]:
```
from sqlalchemy.orm import relationship
```

Creating Our Class Mappings

Now that we've created a `Base` object, we can define our classes!

In order to set up our classes, we'll need to define:

- The `__tablename__` for each class
- The attributes of each class, which will be `Column` objects
- The `relationship` that each class has to other classes

Although we haven't explicitly covered how to create relationships, it's not hard--just a single line of code. This is a great opportunity to get some practice finding what you need from documentation, and the SQLAlchemy documentation is really informative and easy to understand.

We'll be creating a 1-to-1 relationship (Customer <--> ShoppingCart), and a 1-to-many relationship (ShoppingCart <--> Item). Take a look at the documenation for creating relationships and see if you can figure out how to set this up!

In the cell below:

- Complete the `Customer`, `ShoppingCart`, and `Item` classes.
- Give each class the correct table name ('customer', 'shoppingCart', and 'item')
- Define the correct columns for each class, with the appropriate data types, and set the appropriate primary key and foreign keys.
- Set the appropriate relationships between classes.

Hint: When setting the relationships, pay attention to the capitalization in the documentation--in some parts, you reference the name of the class, while in others, you reference the name of the table!

Note: Running a cell more than one time will cause a "Table is already defined" error. To fix this, just restart the kernel and run everything again.

In [3]:
```
class Customer(Base):
    __tablename__ = 'customer'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    cart_id = Column(Integer, ForeignKey('shoppingCart.id'))

    # Create 1-to-1 relationship with ShoppingCart, as shown in the SQLAlchemy documentation
    shoppingCart = relationship('ShoppingCart', uselist=False, back_populates='customer')
```

In [4]:
```
class ShoppingCart(Base):
    __tablename__ = "shoppingCart"

    id = Column(Integer, primary_key=True)
    item_id = Column(Integer, ForeignKey('item.id'))
    # Create 1-to-1 relationship with Customer
    customer = relationship('Customer', uselist=False, back_populates='shoppingCart')
    # Create 1-to-many relationship with Item
    items = relationship('Item')
```

In [5]:
```
class Item(Base):
    __tablename__ = 'item'

    id = Column(Integer, primary_key=True)
    description = Column(String)
    price = Column(Float)
```

Creating Our Database

Now that we've successfully defined our mappings, we can actually create our database. We'll call our database `shopping_cart.db`.

In the cell below:

- Create an `engine` using the appropriate method.
- Use the `create_all()` method found inside of `Base.metadata` and pass in the engine object to create our database!

In [6]:
```
engine = create_engine('sqlite:///shopping_cart.db', echo=True)
Base.metadata.create_all(engine)
```

```
2018-10-30 16:21:37,961 INFO sqlalchemy.engine.base.Engine SELECT CAST('test plain returns' AS VARCHAR(60)) AS anon_1
2018-10-30 16:21:37,962 INFO sqlalchemy.engine.base.Engine ()
2018-10-30 16:21:37,964 INFO sqlalchemy.engine.base.Engine SELECT CAST('test unicode returns' AS VARCHAR(60)) AS anon_1
2018-10-30 16:21:37,965 INFO sqlalchemy.engine.base.Engine ()
2018-10-30 16:21:37,967 INFO sqlalchemy.engine.base.Engine PRAGMA table_info("customer")
2018-10-30 16:21:37,967 INFO sqlalchemy.engine.base.Engine ()
2018-10-30 16:21:37,969 INFO sqlalchemy.engine.base.Engine PRAGMA table_info("shoppingCart")
2018-10-30 16:21:37,970 INFO sqlalchemy.engine.base.Engine ()
2018-10-30 16:21:37,971 INFO sqlalchemy.engine.base.Engine PRAGMA table_info("item")
2018-10-30 16:21:37,971 INFO sqlalchemy.engine.base.Engine ()
2018-10-30 16:21:37,973 INFO sqlalchemy.engine.base.Engine
CREATE TABLE item (
        id INTEGER NOT NULL,
        description VARCHAR,
        price FLOAT,
        PRIMARY KEY (id)
)


2018-10-30 16:21:37,974 INFO sqlalchemy.engine.base.Engine ()
2018-10-30 16:21:37,977 INFO sqlalchemy.engine.base.Engine COMMIT
2018-10-30 16:21:37,979 INFO sqlalchemy.engine.base.Engine
CREATE TABLE "shoppingCart" (
        id INTEGER NOT NULL,
        item_id INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(item_id) REFERENCES item (id)
)


2018-10-30 16:21:37,985 INFO sqlalchemy.engine.base.Engine ()
2018-10-30 16:21:37,987 INFO sqlalchemy.engine.base.Engine COMMIT
2018-10-30 16:21:37,988 INFO sqlalchemy.engine.base.Engine
CREATE TABLE customer (
        id INTEGER NOT NULL,
        name VARCHAR,
        cart_id INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(cart_id) REFERENCES "shoppingCart" (id)
)


2018-10-30 16:21:37,989 INFO sqlalchemy.engine.base.Engine ()
2018-10-30 16:21:37,991 INFO sqlalchemy.engine.base.Engine COMMIT
```

CRUD Operations

We've now created a database, but our tables don't contain any data yet!

We'll need to create some objects, and then populate the database with them.

Run the cell below to some sample data for our tables.

In [7]:
```
customer1 = Customer(name="Jane")
item1 = Item(description="widget", price=9.99)
cart1 = ShoppingCart(customer=customer1, items = item1)
customer1.shoppingCart = cart1
```

Note that this data has not yet been put into the database. Before that happens, we need to create a `session` object, then add these objects and commit them. We can double check this by examining the items and seeing that they don't yet have primary keys. Run the cell below now to demonstrate this.

In [8]:
```
customer1.id, item1.id
```

Out[8]: (None, None)

You may have noticed that we defined values for certain attributes such as the customer's name, or the item's description and price, but never attributes that act as ids. There's a reason for this--SQLAlchemy takes care of this for us! Since every primary key has to be unique, this means that defining the integer values for primary keys would be really cumbersome, since we would need to keep track of every primary key that's been created so far--a much better task for a computer than for us!

Another thing you might have noticed is that to create relationships between objects, we just assign them to attributes that were defined as `relationship` objects when we created our mappings!

Creating a Session Object

In order to add our new data to our database tables, we first need to create a session object.

In the cell below:

- import `Session` and `sessionmaker` from `sqlalchemy.orm`
- create a `sessionmaker` and set the `bind=` parameter to our `engine` object. Store this in `Session`
- Instantiate a `Session()` object and store it in the variable `session`

In [9]:
```
from sqlalchemy.orm import sessionmaker, Session
Session = sessionmaker(bind=engine)

session = Session()
```

Great! Now we have a session object that we can use to interact with our database.

We can add items to our database one at a time by passing them in as a parameter to `session.add()`. We can also add multiple items by passing them as a list into the `add_all()` method. In the cell below, use `add_all()` to add `customer1`, `cart1`, and `item1` into our database.

In [10]:
```
session.add_all([customer1, cart1, item1])
```

Adding something multiple times will not throw an error or cause duplicates. We can see all the items that have been added by checking the session object's `.new` attribute. Do this now in the cell below.

In [11]:
```
session.new
```

Out[11]:
```
IdentitySet([<__main__.Customer object at 0x108e90a58>, <__main__.ShoppingCart object at 0x108eac6d8>, <__main__.Item object at 0x108e90668>])
```

Now, commit our objects to push them to the database.

In the cell below, call `session.commit()`.

In [12]:
```
session.commit()
```

```
2018-10-30 16:21:38,058 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
2018-10-30 16:21:38,061 INFO sqlalchemy.engine.base.Engine INSERT INTO item (description, price) VALUES (?, ?)
2018-10-30 16:21:38,061 INFO sqlalchemy.engine.base.Engine ('widget', 9.99)
2018-10-30 16:21:38,063 INFO sqlalchemy.engine.base.Engine INSERT INTO "shoppingCart" (item_id) VALUES (?)
2018-10-30 16:21:38,064 INFO sqlalchemy.engine.base.Engine (1,)
2018-10-30 16:21:38,066 INFO sqlalchemy.engine.base.Engine INSERT INTO customer (name, cart_id) VALUES (?, ?)
2018-10-30 16:21:38,067 INFO sqlalchemy.engine.base.Engine ('Jane', 1)
2018-10-30 16:21:38,069 INFO sqlalchemy.engine.base.Engine COMMIT
```

If we check the object ids again, we'll see that they now have values for their primary keys.

In the cell below, check the `.id` attribute of `customer1`.

In [13]:
```
item1.id
```

```
2018-10-30 16:21:38,080 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
2018-10-30 16:21:38,084 INFO sqlalchemy.engine.base.Engine SELECT item.id AS item_id, item.description AS item_description,
item.price AS item_price
FROM item
WHERE item.id = ?
2018-10-30 16:21:38,085 INFO sqlalchemy.engine.base.Engine (1,)
```

Out[13]:
```
1
```

Summary

In this lab, we created a database with SQLAlchemy, defined our mappings to structure the tables, and even added some data to the database. Great job!