Insertion

In this lesson, we will learn how to populate our database by inserting data.

In the last chapter, we figured out how to connect to a database with the Flask application. Furthermore, we learned how to create `Models` and relationships between these models. In this chapter, we will be focusing on all kinds of operations that we can perform on our models and the changes that will occur in the database. The type of operations that we will need to make are:

1. **Insert**
2. **Delete**
3. **Retrieve**
4. **Update**

Introduction

First, we need to populate our database with the data. Let's insert some data using the `User` model that we created in the last chapter.

```
class User(db.Model):
    email = db.Column(db.String, primary_key=True, unique=True, nullable=F
alse)
    password = db.Column(db.String, nullable=False)
```

1. Create a new `User` object

To create a new user we will simply create an object of the `User` class.

```
new_user = User(email = "archie.andrews@email.com", password = "football4life")
```

2. Add the `new_user` to the database

We will then add this user to the current `session` of the `SQLAlchemy` instance (i.e., `db`).

> **Note:** the `session` variable indicates an ongoing transaction of changes to the database. It keeps a record of **insertion**, **updates**, and **deletion**.

```
db.session.add(new_user)
```

3. `commit()` the changes in the database

To finish off, these changes will be committed to the database using the `commit()` function.

> **Note:** without committing the changes, the updated state will not persist in the database. It will only remain persistent in the current `session`. In other words, if we do not commit these changes, but try to retrieve this object in the current session, we will be successful. But, when this session closes, everything we inserted/updated will be lost.

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

Complete implementation

Let's create an example script that does not contain any views or templates so that we can focus on the models. The above steps are present in the script below.

Note: this script is not a web application. It is a simple Python program that runs and shows the output on the console. At the end of this chapter, we will incorporate these steps into a web application as well.

```
from flask import Flask, render_template
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class User(db.Model):
    email = db.Column(db.String, primary_key=True, unique=True, nullable=False)
    password = db.Column(db.String, nullable=False)

db.create_all()

archie = User(email = "archie.andrews@email.com", password = "football4life")
veronica = User(email = "veronica.lodge@email.com", password = "fashiondiva")

db.session.add(archie)
db.session.add(veronica)

try:
    db.session.commit()
except Exception as e:
    db.session.rollback()
finally:
    db.session.close()
```

Explanation

1. In **lines 15 - 16**, we created the object `Users`.
2. In **lines 18 - 19**, we placed these objects inside the session.
3. Finally, in **line 22**, we committed these changes to the database.
4. If anything goes wrong while committing the changes, then the `commit()` function will throw an exception. To catch this `Exception` we have used a `try-except` block.

5. In the case of an `Exception`, we will `rollback()` the changes in **line 24**.

6. Finally, in **line 26**, we close the current database `session` by calling `db.session.close()`.

In the next lesson, we will learn how to retrieve data we just inserted.