Database Connection using SQLAlchemy

In this lesson, we will be exploring how to create a database connection in your Flask application by using SQLAlchemy.

WE'LL COVER THE FOLLOWING

Introduction

Finally, in this chapter, we will learn to handle the data using a database. Up until now, we have been using some data structures like a dictionary, but in a real-world application, this approach is *rarely*, **if ever**, used. We will be using an SQL database and an object relation mapper to manipulate that database inside `Flask`.

What is an ORM or object relation mapper?

An **ORM** makes writing SQL queries easier for a programmer. It enables us to write queries in an object-oriented language, and then the **ORM** automatically translates it to SQL and retrieves the results in the form of objects!

> ### Introduction to `SQLAlchemy`
>
> `SQLAlchemy` is a library in Python which allows us to manipulate SQL. It provides us with an easy to use `ORM` for SQL databases.

> ### Introduction to `Flask-SQLAlchemy`
>
> `Flask-SQLAlchemy` is a `Flask` specific library that integrates the `SQLAlchemy` support with `Flask` applications. It provides extra helpers for common tasks that make it easier to work with `Flask`.

How to initiate a database connection

That's enough about theoretical concepts. Let's dive deep into the programming aspect of databases. Initially, we must first create a database connection. We take the following steps to initiate the database connection:

Import `SQLAlchemy`

We will first import the `SQLAlchemy` class from the `flask_sqlalchemy` module in the main application file of our project, i.e., `app.py`.

```
from flask_sqlalchemy import SQLAlchemy
```

Set the `config` variable to the database file

We will have to set a configuration variable in the application so that the application knows where the database file is located. The SQL database we are using is `SQLite`. Therefore, the file name will contain the prefix `sqlite:///` followed by the actual path of the file. However, you can use any other SQL database with SQLAlchemy as well. Now, we will set the config variable `SQLALCHEMY_DATABASE_URI` to point to this file.

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
```

Initialize the database connection

Now, to complete the initialization, we just have to create an object of the `SQLAlchemy` class. We have to provide our application as a parameter to its constructor.

```
database = SQLAlchemy(app)
```

Complete implementation

Let’s integrate the steps we discussed in the example we covered in the last chapter.

```
#header {
  padding: 30px;
  text-align: center;
  background: #140005;
  color: white;
  font-size: 40px;
}
#footer {
   position: fixed;
   width: 100%;
   background-color: #BBC4C2;
   color: white;
   text-align: center;
   left: 0;
   bottom:0;
}
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

li {
  display: inline;
}
```

Now that we know how to connect to the database, in the next lesson, we will learn how to create tables in the database using models. Stay tuned!