Database Layer CRUD

Contents

Overview

In this module we will look at how to perform CRUD operations with your database using Flask and SQLAlchemy.

Operations

To be able to perform CRUD operations you must have instantiated a `db` object and defined your schema classes.

For our examples we will use the game database which is defined as:

```
db = SQLAlchemy(app)

class Games(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, unique=True)
```

Create

To add a new entry into a database using SQLAlchemy you need to first instantiate an object which is of your table's class type.

For example, to create a game entity to insert into our `Games` table we'd write:

```
new_game = Games(name="name_of_new_game")
```

> Note: Despite there being an `id` column defined in our schema, we do not need to specify is as an argument as it is the primary key and will increment automatically.

Now we have our new game instantiated as the `new_game` object in our Python code. In order to add it to the database, we first use the method `db.session.add()`:

```
db.session.add(new_game)
```

We now have added the new game to the db object. This only *stages* the new entity for addition to the database; the information has not yet been inserted into the table.

To commit our staged addition, we run:

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

With this method of staging and committing changes to the database, we can stage multiple additions and commit them all to the database with one command.

Read

In order to read information from a database, we need to write queries. Queries are made with methods that belong to class of the table you are querying.

There are a number of ways to query a database with SQLAlchemy that change the form of the data you receive.

All

To retrieve all the entries from a given table, we write:

```
Games.query.all()
```

This will return a list of all entries in the `Games` as objects of type `Games`.

> Note: Even if there's only one entry retrieved from the database, it will *still* be returned as a list.

You can store the response to this function in a variable.

```
all_games = Games.query.all()
```

First

You can also just query for the first entry in any table.

```
first_game = Games.query.first()
```

As this will only retrieve one, the query will return a single object of the table's class type. This then allows you to access the entity's fields as a class attribute. For example:

```
first_game = Games.query.first()

print(first_game.name) # would print the name of the first game in the table

print(first_game.id) # would print the id of the first game
```

> Note: Either the `first()` or `all()` method is required at the end of a given queries.

Filter By

If you only need to retrieve *some* of the available entries from a table, you can use `filter_by()` to retrieve only relevant information.

This method expects one or more of the table's columns as parameters. In this example we are retrieving the entity with the `id` value of `1`.

```
game_with_id_1 = Games.query.filter_by(id=1).first()
```

In this example, we filter by the `name` column for all games with the name 'Dark Souls'.

```
dark_souls = Games.query.filter_by(name="Dark Souls").all()
```

Get

You can also retrieve an entry by the primary key using `get`.

```
game_with_id_1 = Games.query.get(1)
```

This is the equivalent of writing `Games.query.filter_by(id=1).first()` but is more readable.

Order By

You can change the order your information is retrieved in by using the `order_by()` method.

For example:

```
games_in_order = Games.query.order_by(Games.name).all()
```

Here we are ordering the list of games we get from the database alphabetically by the games' names. We do this by passing the column attribute belonging to the table's class.

You can also add `.desc()` on the end of your `order_by()` parameter to put the entries in descending order.

```
games_in_order = Games.query.order_by(Games.name.desc()).all()
```

Limit

If you only want a certain number of entries in your query, you can use `limit()`.

This function takes an integer as a parameter and will only retrieve that many entries.

```
first_2_games = Games.query.limit(2).all()
```

Will return the first 2 games.

Count

You can use the `count()` method to return the number of entries that match a given query, rather than the entries themselves.

```
number_of_games = Games.query.count()
```

This would give the number of games as an **integer**.

Update

Let's look at how to update existing entries in the database.

In order to update an entry, we first need to instantiate it as a Python object. We can do this by querying the database:

```
first_game = Games.query.first()
```

Then we can assign new values to the object's attributes:

```
first_game.name = "New name"
```

Finally, we need to commit this change to the database.

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

Delete

In order to delete objects from a database, we first need to retrieve and instantiate it within the Python code:

```
game_to_delete = Games.query.first()
```

Then we can stage the deletion of the entry with the `db` object like so:

```
db.session.delete(game_to_delete)
```

And finally, commit the staged deletion to the database:

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

Tutorial

For this tutorial we are going to create an app that performs CRUD operations on a database table.

Feel free to use any database layer you like whether that be an inbuilt `sqlite` or a stand-alone `mysql` server.

Setup

This tutorial assumes you are working on an Ubuntu VM, at least version `18.04 LTS`.

First, install apt dependencies:

```
sudo apt update
sudo apt install python3 python3-venv python3-pip
```

Create a directory named `flask-db-crud` and make it your current working directory:

```
mkdir flask-db-crud && cd $_
```

We now need to create a Python virtual environment to install our pip requirements in. Create a new virtual environment named `venv` and activate it:

```
python3 -m venv venv
source venv/bin/activate
```

File Structure

This is the structure of the application we are going to create:

```
.
├── application
│   ├── __init__.py
│   ├── routes.py
│   └── models.py
├── create.py
├── requirements.txt
└── app.py
```

To create this file structure, run:

```
mkdir application
touch application/{__init__,routes,models}.py create.py app.py requirements.txt
```

Paste the following into `requirements.txt`:

```
flask
flask_sqlalchemy
pymysql
```

This is the list of pip dependencies that the app requires in order to run. Run the command to install them:

```
pip3 install -r requirements.txt
```

Create the App

Paste the following into `app.py`:

```
from application import app

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0')
```

Paste the following into `create.py`:

```
from application import db

db.drop_all()
db.create_all()
```

Paste the following into `__init__.py`:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///data.db"

db = SQLAlchemy(app)

from application import routes
```

Paste the following into `models.py`:

```
from application import db

class Games(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(30))
```

Paste the following into `routes.py`:

```
from application import app, db
from application.models import Games

@app.route('/add')
def add():
    new_game = Games(name="New Game")
    db.session.add(new_game)
    db.session.commit()
    return "Added new game to database"

@app.route('/read')
def read():
    all_games = Games.query.all()
    games_string = ""
    for game in all_games:
        games_string += "<br>"+ game.name
    return games_string

@app.route('/update/<name>')
def update(name):
    first_game = Games.query.first()
    first_game.name = name
    db.session.commit()
    return first_game.name
```

Run the App

Now create your database's schema by running:

```
python3 create.py
```

And run the app with the command:

```
python3 app.py
```

Access the app in your browser via your machine's public IP address on port `5000` and do the following:

1. Go to the `/add` route the application will add a new game to the database.
2. Got to `/read` you will see all the games in the database.
3. Go to `/update/<name>`, replacing `<name>` with any name you like, to update the name of the first entry in the games table.

Clean Up

To stop your Flask application running, navigate back to your terminal and press `Ctrl+C`. You should now have control over your terminal again.

To deactivate the virtual environment, run:

```
deactivate
```

If you wish to delete the virtual environment, run:

```
rm -rf venv
```

Exercises

Add a new route called `/delete` that will delete the first entry in the database. Have it return a message to let you know you have deleted the entry.

Add another route that returns the number of games in the database.