

launch _code

LC101 2.9



Class Agenda

1. Last Class Review
2. Announcement (Geek Gala)
3. Studio



Announcement

GeekGala - LaunchCode's birthday party is happening on October 19th!

If you are interested in attending please refer to the announcement on canvas for more information.



Review

SQL DB Basics - CREATE TABLE, DROP TABLE

SQL Table Basics - SELECT, INSERT INTO, DELETE, UPDATE

SQL Relationships - Primary Key, Foreign Key, One to Many

phpMyAdmin - our portal into MySQL



ORM

ORM - Object Relational Mapping describes the process of taking an object from an application translating and storing it in a relational database

It gives you a very easy way to bind an application with a DB. Many ORM's will handle a lot of the logic and code for you. They also keep you orgainzed.



ORM continued

We will be using SQL Alchemy as our ORM for Unit 2. Documentation:

<http://docs.sqlalchemy.org/en/latest/>

There are many different ORM's, in unit 3 we will use either Hibernate (Java), or the Entity Framework's ORM (C#).

The syntax will be different, but the concepts will be the same.



ORM continued

ORM's are based on mapping Objects (classes) from your application to the relational database (MySQL).

If you need a refresher on classes check out chapters 12 & 13 of Think Python from Unit 1.

We are essentially going to create a Class that mimics the tables in our DB. And we create a DB object that performs the SQL for us. By calling functions in the library it queries our DB and returns to us the results.



Working with an ORM

1. Create your DB, and a user for your DB.
(lecture)
2. Create a Flask app (lecture)
3. Configure the DB in your Flask app (lecture)
4. Create your classes that map to your DB
(lecture)
5. Create the logic in your route (studio)
6. Create the templates (studio)



Prep Work Walkthrough

Let's look at an example.



Project Setup

We need to be in a conda environment with SQLAlchemy and pymysql installed.

We need to create a normal hello world flask app, as we normally would.



DB Configuration

Create a new user, and DB of the same name.

We are going to take note of username, password, server address (localhost), port, DB name

In our flask app we are going to need to create a new DB from SQLAlchemy.

In our flask app, we are going to change some DB configurations so that our flask app knows where our DB is located.

Persistent Classes in Flask

Data Persistence - The data is preserved even when the application is not currently running.

Build a class that reflects the tables of our DB.

These will be unique to the tables you need for your application to store your data.

In our example it was creating the Task class.



Task Class

We can add now access our DB from python using the terminal, and create new tasks, then committing the tasks to our DB.

We can now write some additional python code that will activate on a server call that will automatically write to our DB.




Relationships

The power of SQL is in the relationships between tables, so we do need to talk about how that is handled in Flask.

In python we mimic what we did in SQL. We need to link the two classes together by adding a new parameter that contains a foreign key to the other table.

This requires us to make some changes to the classes of linked tables, changes some of the logic of our route, and sometimes changes our templates



User Class

To complete our prep work we walked through creating an entirely new class, that had some extra features. The user needs to be able to login. I am not walking through that today, because I want you to focus on ORM, not handling users.

As you need Users in your projects, please refer to the prep work to walk through the steps again.



Studio - Flicklist 6

Walkthrough6a - creating a DB, configuring it with our flask app, and accessing it from the command line in python

Walkthrough6b - changing `get_current_watchlist()` to return Movie objects, we also change our templates

Studio6 - add some changes to the routing logic that will make our changes persistent

