Databases, Queries, and Joins

No screens

Prof. Lydia Chilton
COMS 6998
14 September 2018
You already know **front-end web dev:** HTML, JavaScript, Bootstrap, jQuery

And design:
Iterative design, critique

You will learn back-end web dev:
- **Server-side programming (Flask),**
- Databases (Sqlite, SQLAlchemy)
- Real-time Communication (Socket.IO)

And practice web design by:
- **Rebuilding IMDB.com**
- Rebuilding twitter
- Pursuing your own project
Rebuilding IMDB.com
What is the single concrete user goal that best defines IMDB?
Who is that actor??????
What you just turned in:
Remake the **basic functionality** of IMDB.com

- Must use Flask (web server) back end
- HTML, JavaScript, jQuery, Bootstrap
- Must have multiple pages
- Must serve data from the server

- **NO** static HTML data to display data
- **NO** Database
- **NO** images
- **NO** graphic design
To remake the **basic functionality** of IMDB.com

What goals should we define?

- **High-level user goal:**
  - ???

- **Low-level dev goals:**
  - ??
  - ??
  - ??
  - ??
  - ??
  - ??
This is a **studio class**.
We practice web dev and learn from experience.

Like this:

Not this:
Studio Time

Count off by 3’s
What to discuss during studio

• What was your high level goal?
  • (Get at least two answers)

• Show us your homepage.
  • (Get at least two answers)

• What pages did you create?
  • (Get at least two answers)

• Let’s see the code for the route.
  • (Get at least two answers)

• What does your ”database” look like?
  • (Get at least two answers)

• How does the user navigate your site to accomplish the goals multiple times?
  • (Get at least two answers)

• What did you discover or learn?
  • (Everybody answer)

• What are the pros and cons of this database implementation?
My High-level goal

Allow a user to look up the main actors in a movie.

I learned that storing the actors in a movie and the movies of an actor was redundant. So I changed the goal.

Allow a user to look up the year a movie was made (or actor was born).
My Homepage

HOME

Crazy Rich Asians: 2018  Delete 1
Mission Impossible: Fallout: 2017  Delete 2
Jurassic Park: 1993  Delete 3
The Big Lebowski: 1998  Delete 4

Add Movie
Henry Golding
Constance Wu
Tom Cruise

FOOTER
What did I learn

• I spent a lot of time restarting the server.
• I learned about Flask **debug mode**.
• Restarts the server automatically on code changes (still have to reload the page)
What did I learn

• I was able to add new movies to the “database”
• But I don’t give any feedback. I added that to my low level goals, but didn’t do it.
Studio discussion: (30 minutes)

• What was your high level goal?
  • (Get at least two answers)

• Show us your homepage.
  • (Get at least two answers)

• What pages did you create?
  • (Get at least two answers)

• Let’s see the code for the route.
  • (Get at least two answers)

• What does your ”database” look like?
  • (Get at least two answers)

• How does the user navigate your site to accomplish the goals multiple times?
  • (Get at least two answers)

• What did you discover or learn?
  • (Everybody answer)

• What are the pros and cons of this database implementation?

At the end of studio, make a public piazza post saying something you learned.
This week’s high level goal:

• Remake the basic functionality of IMDB.com:
  • User goal: “Look up what actor was in what movie and then see all the other movies that actor is in.”

• Must store data in a SQLite database
• Must query it with SQLAlchemy
• Must Enable CRUD operations (create, read, update, delete)
  • This is not actually user-facing functionality of IMDB, but it’s essential back-end dev)
• Must use at least one Database Join
This week’s high level goal:

• Remake the basic functionality of IMDB.com:
  • User goal: “Look up what actor was in what movie and then see all the other movies that actor is in.”

• Must store data in a SQLite database
• Must query it with SQLAlchemy
• Must Enable CRUD operations (create, read, update, delete)
  • This is not actually user-facing functionality of IMDB, but it’s essential back-end dev)
• Must use at least one Database Join
Dictionaries vs. Database Tables: Which is faster to query?

movies = {
    '1': {
        'title': "Crazy Rich Asians",
        'rating': "PG-13",
    },
    '2': {
        'title': "Mission Impossible: Fallout",
        'rating': "G",
    }
}

actors = {
    '1': {
        'name': "Henry Golding",
        'year_of_birth': "1987",
    },
}

"Movie" table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
</tr>
</tbody>
</table>

"Actor" table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
</tr>
</tbody>
</table>
How did IMDB do this so fast???
Dictionaries vs. Database Tables: How do we find all movies named ‘Titanic’?

movies = {
    '1': {
        'title': "Crazy Rich Asians",
        'rating': "PG-13",
    },
    '2': {
        'title': "Mission Impossible: Fallout",
        'rating': "G",
    }
}

actors = {
    '1': {
        'name': "Henry Golding",
        'year_of_birth': "1987",
    },
}

"Movie" table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
</tr>
</tbody>
</table>

"Actor" table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
</tr>
</tbody>
</table>
For applications with
~100 users
~500 data items,
I have deployed dictionaries!

Great for prototyping!
But the data is not persistent.
(What does that mean?)
This week’s high level goal:

• Remake the basic functionality of IMDB.com:
  • User goal: “Look up what actor was in what movie and then see all the other movies that actor is in.”

• Must store data in a **SQLite database**
• Must query it with **SQLAlchemy**
• Must Enable **CRUD** operations (create, read, update, delete)
  • This is not actually user-facing functionality of IMDB, but it’s essential back-end dev)
• Must use at least one **Database Join**
Before SQLAlchemy, there was SQL

"Movies" table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
</tr>
</tbody>
</table>

We want to be able to query the database. For example, find a movie title by its id:

"SELECT title FROM Movies WHERE id = 1"

Problem: How do you write SQL statements in the server language (python, PHP)?
Writing SQL in PHP:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
$id = $_GET["id"]
$sql = "SELECT title, year FROM Movies WHERE id = $id";
$result = $conn->query($sql);

echo $result
$conn->close();
?>
```
Writing SQL in PHP:

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
$id = "1; DROP TABLES;"
$sql = "SELECT title, year FROM Movies WHERE id = $id";
$result = $conn->query($sql);

echo $result
$conn->close();
?>
```
from app import db
from app.models import Movie, Actor

@app.route("/movie/<id>")
def movie(id):
    movie = Movie.query.get(id)
    return render_template('movie.html', movie_data = movie)
This week’s high level goal:

• Remake the basic functionality of IMDB.com:
  • User goal: “Look up what actor was in what movie and then see all the other movies that actor is in.”
  • Must store data in a SQLite database
  • Must query it with SQLAlchemy
  • Must Enable CRUD operations (create, read, update, delete)
    • This is not actually user-facing functionality of IMDB, but it’s essential back-end dev)
  • Must use at least one Database Join
How will we keep track of what actors are in what movies?

“Movie” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
</tr>
</tbody>
</table>

“Actor” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
</tr>
</tbody>
</table>
Can we add an Actors column to the Movie table?

“Movie” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
<th>actors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
<td>Henry Golding, Constance Wu</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
<td>Tom Cruise, Alec Baldwin, Superman</td>
</tr>
</tbody>
</table>

“Actor” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
</tr>
</tbody>
</table>
Can we add an actors column to the Movie table AND a movies column to the Actor table?

“Movie” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
<th>actors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
<td>Henry Golding, Constance Wu</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
<td>Tom Cruise, Alec Baldwin, Superman</td>
</tr>
</tbody>
</table>

“Actor” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
<th>Movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
<td>Crazy Rich Asians</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
<td>Crazy Rich Asians</td>
</tr>
<tr>
<td>3</td>
<td>Tom Cruise</td>
<td>1962</td>
<td>MI 1, MI 2, MI 3, MI4, MI5, MI: Fallout</td>
</tr>
</tbody>
</table>
Can we add an actors column to the Movie table AND a movies column to the Actor table?

“Movie” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
<th>actors</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
<td>1, 2</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
<td>3</td>
</tr>
</tbody>
</table>

“Actor” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
<th>Movies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Tom Cruise</td>
<td>1962</td>
<td>2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12</td>
</tr>
</tbody>
</table>
Join tables (also called Association tables)

```
<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
</tr>
</tbody>
</table>

“Actor” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
</tr>
<tr>
<td>3</td>
<td>Tom Cruise</td>
<td>1962</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Movie_id</th>
<th>Actor_id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Movie_id</th>
<th>Actor_id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Join tables (also called Association tables)

**“Movie” table**

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
</tr>
</tbody>
</table>

**“Actor” table**

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
</tr>
<tr>
<td>3</td>
<td>Tom Cruise</td>
<td>1962</td>
</tr>
</tbody>
</table>

**Movie_actor_join**

<table>
<thead>
<tr>
<th>Movie_id</th>
<th>Actor_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
Now querying for actors and movies is easy

“Movie” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>year</th>
<th>Actors_backref</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crazy Rich Asians</td>
<td>PG-13</td>
<td>2018</td>
<td>(automatic)</td>
</tr>
<tr>
<td>2</td>
<td>MI: Fallout</td>
<td>G</td>
<td>2018</td>
<td></td>
</tr>
</tbody>
</table>

“Actor” table

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>Year_of_birth</th>
<th>Movie_backref</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Henry Golding</td>
<td>1987</td>
<td>(automatic)</td>
</tr>
<tr>
<td>2</td>
<td>Constance Wu</td>
<td>1982</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Tom Cruise</td>
<td>1962</td>
<td></td>
</tr>
</tbody>
</table>

`Actors.query.get(3).movie_backref`  
`Movies.query.get(1).actors_backref`
Great video on SQLAlchemy Association Tables

Creating Many-To-Many Relationships in Flask-SQLAlchemy

Published on Feb 21, 2016

I talk about how to use Flask-SQLAlchemy to create many-to-many relationships between tables.

To get a free Flask cheat sheet, go here: http://prettyprinted.com/sqlalchemy_f...
This week’s high level goal:

• Remake the basic functionality of IMDB.com:
  • User goal: “Look up what actor was in what movie and then see all the other movies that actor is in.”

• Must store data in a SQLite database
• Must query it with SQLAlchemy
• Must Enable CRUD operations (create, read, update, delete)
  • This is not actually user-facing functionality of IMDB, but it’s essential back-end dev
• Must use at least one Database Join
In the near future:

• Make your own site that puts a new spin on IMDB.com
Turn in by 2pm Friday:

1. Establish goals.
   One high level **user goal** and 7-10 low-level goals that will help you accomplish the high level goal.

2. Iteration.
   Report on 3 of the features in your that caused you to iterate on your goals.
   1. My plan was to ____x____.
   2. But I ran into problem ____y____.
   3. And I solved it by doing ____z____.
   
   z = “I added a new sub-goal”, “I changed my high level goal”, “I removed a sub-goal”

   For each of the goals in part 1. Which items you completed?
   Show images to document each item. (either of the UI or code)

   Be prepared to discuss your progress during studio.

   You will be graded on whether or not you did each part.

   Late work cannot be accepted.
   Turn in whatever you have by 2pm to get credit.