Databases, Queries, and Joins

No screens

Prof. Lydia Chilton
COMS 6998
14 September 2018

Say your name

Columbia University

User Interface Design

COMS 4170 · Spring 2018

Home Syllabus Assignments

Part 1 Build websites that suit the needs and abilities of users.
Part 2 When the needs and abilities of users are uncertain, design systems

You already know front-end web dev:

HTML, JavaScript, Bootstrap, jQuery

And design:

Iterative design, critique

Columbia University

Advanced Web Design Studio

COMS 6998 · Fall 2018

Home Syllabus

Goals

1. Master front-end and back-end technologies for making interactive websites.
2. Discover specific user needs by developing a low-level, mechanical model of human behavior.
3. Practice iterative design to meet specific user needs.

INSTRUCTOR

Prof. Lydia Chilton
OH: Tuesdays 4-5, CEPSR 612

Please contact staff through Piazza only

TAS

Katy Gero
OH: TBA, CS OH room

Savvas Petridis
OH: TBA, CS OH room

You will learn back-end web dev:

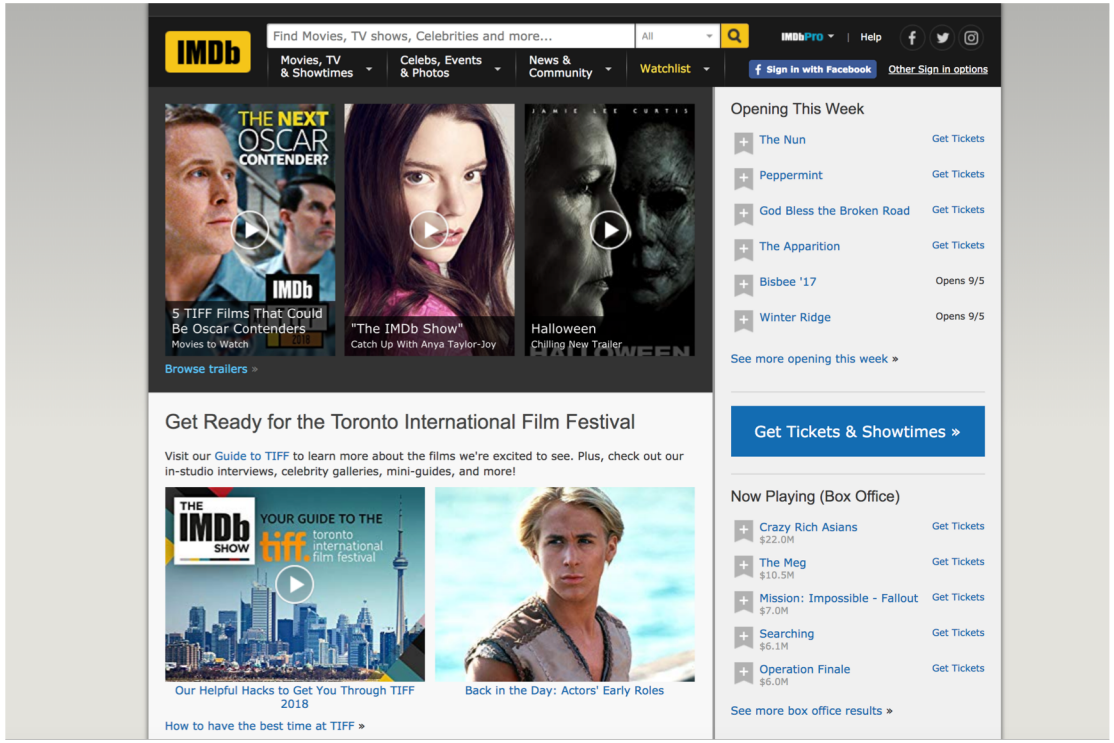
- Server-side programming (Flask),
- Databases (Sqlite, SQLAlchemy)
- Real-time Communication (Socket.IO)

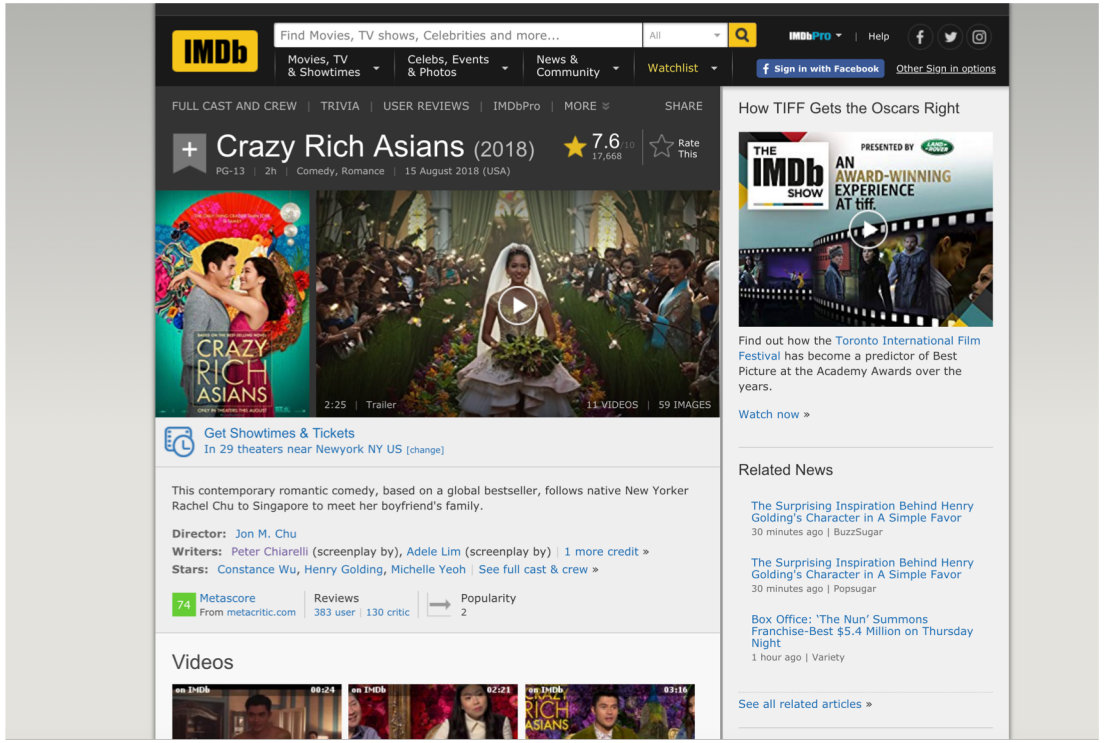
And practice web design by:

- Rebuilding IMDB.com
- Rebuilding twitter
- Pursuing your own project

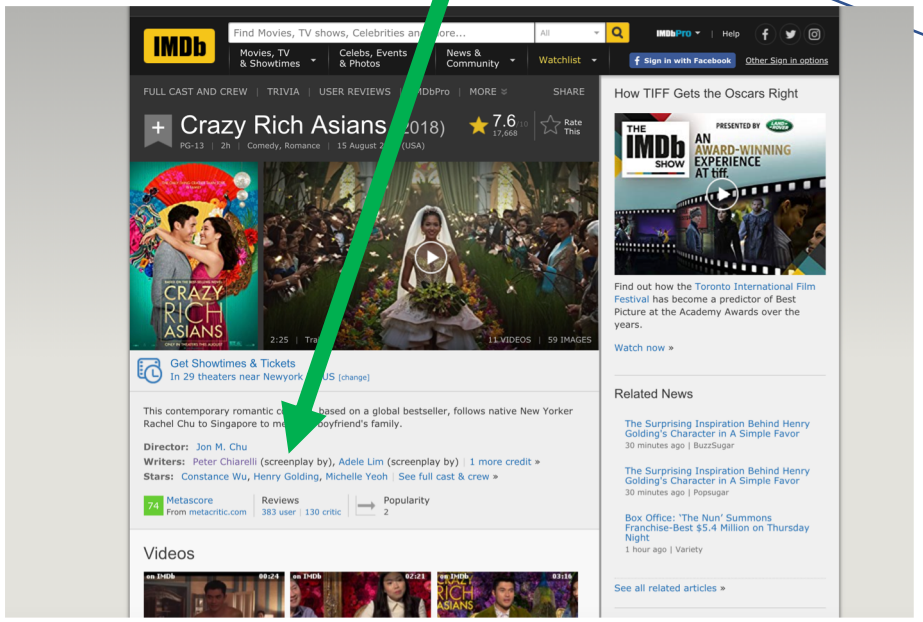
Rebuilding IMDB.com

What is the single concrete user goal that best defines IMDB?





Who is that actor?????



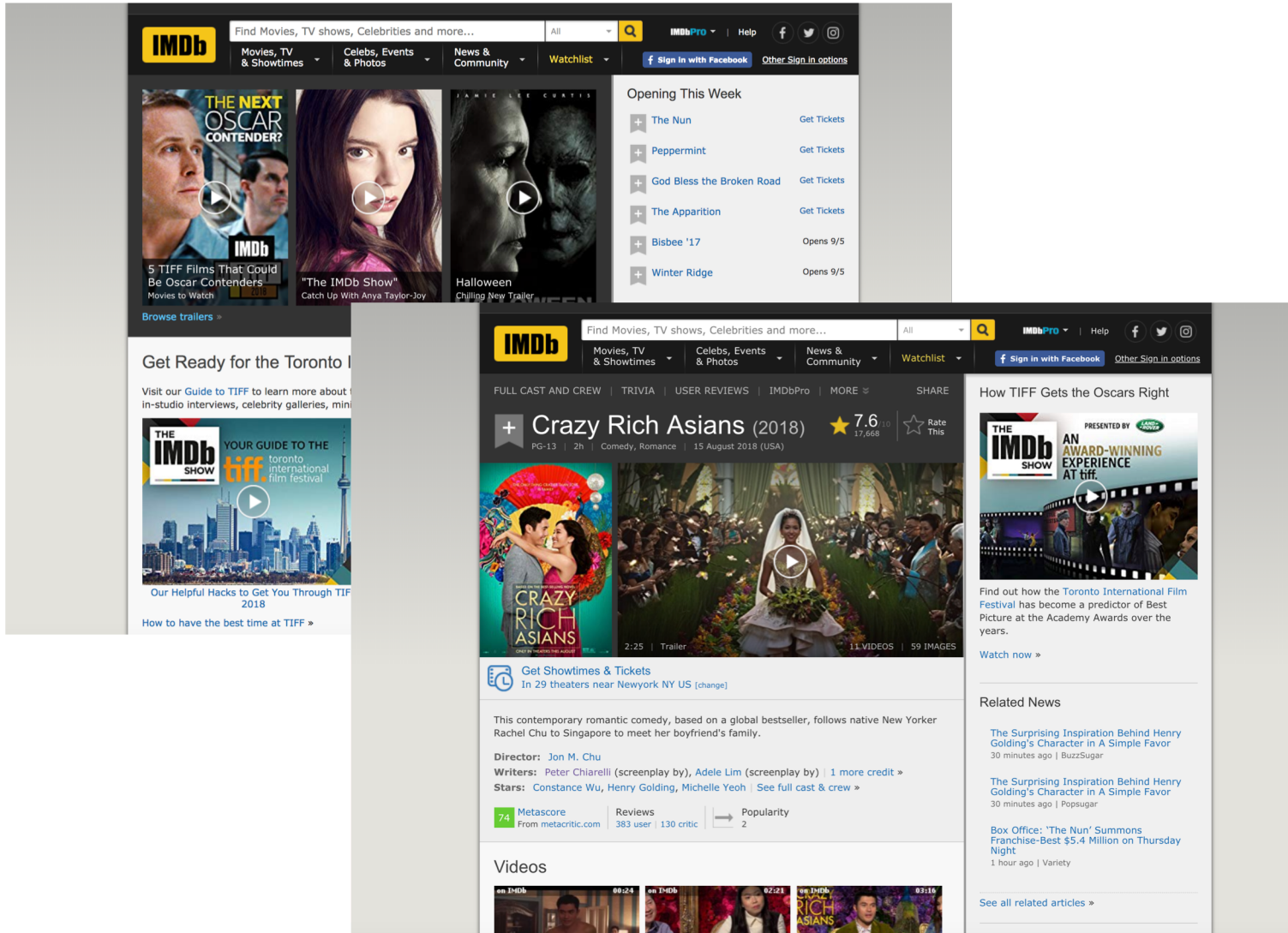
What you just turned in:
Remake the **basic functionality** of IMDB.com

- 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?**

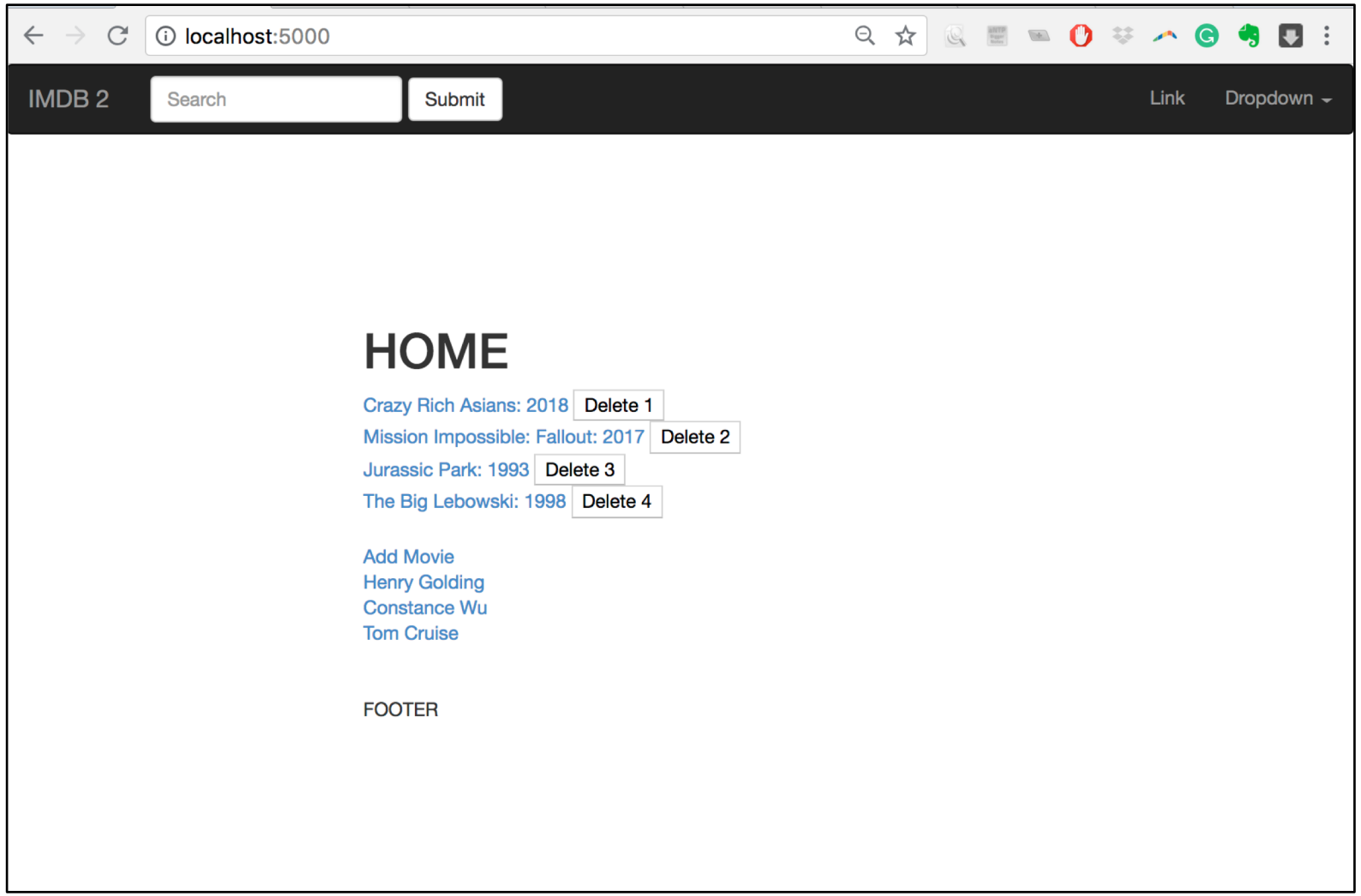
My High-level goal

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


localhost:5000
IMDB 2
Search
Submit
Link
Dropdown
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)

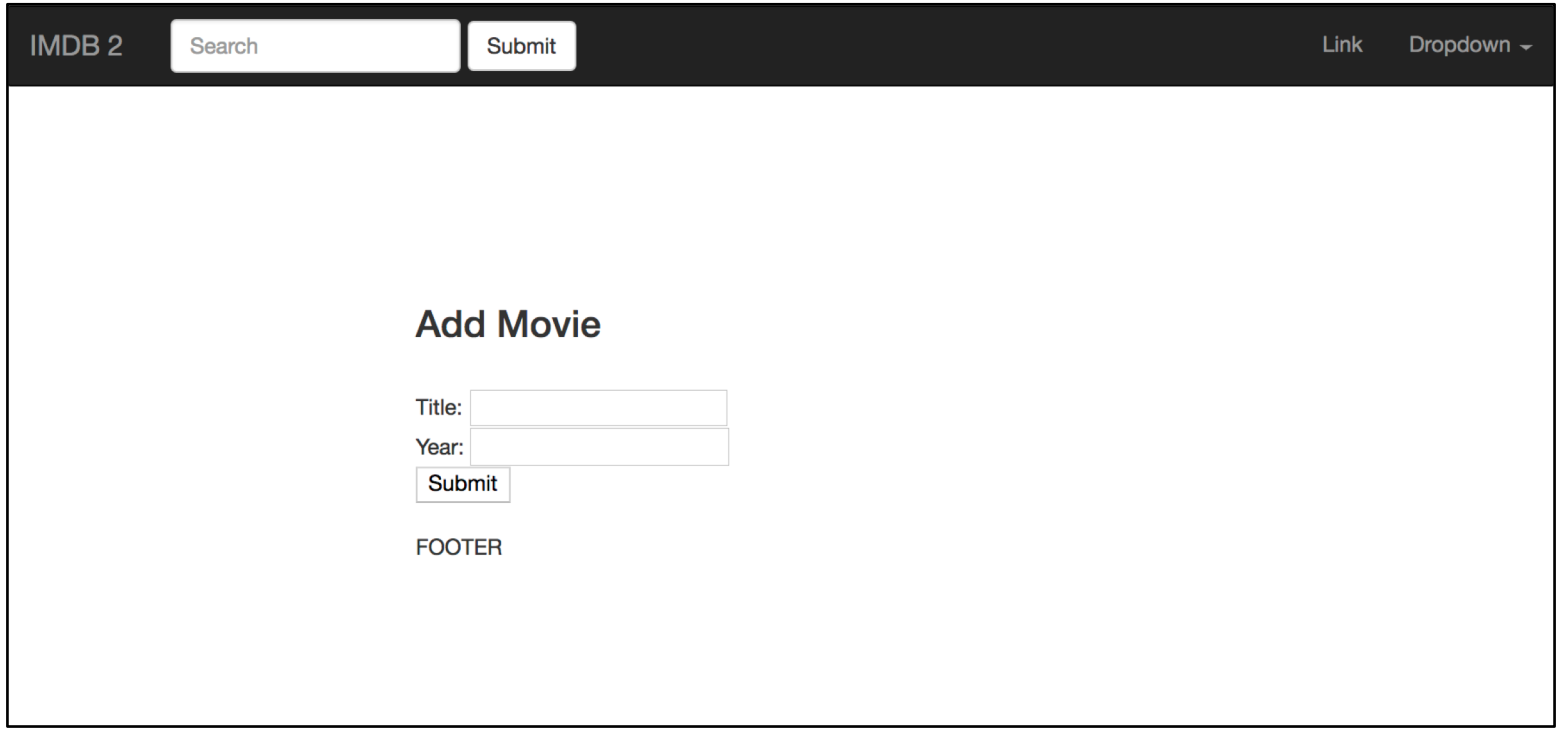
```python
from app import app


if __name__ == "__main__":
    app.run(debug=True)
```

```
imdb — Python ◂ Python app.py — 80×21
...gnStudio/FlaskExperiments/imdb — -bash    ...ents/imdb — Python ◂ Python app.py
Last login: Tue Sep 11 10:08:27 on ttys002
Lydias-MacBook-Pro:imdb lydiachilton$ python3 app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 103-038-942
```

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 funcationality 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",
    },
}
```

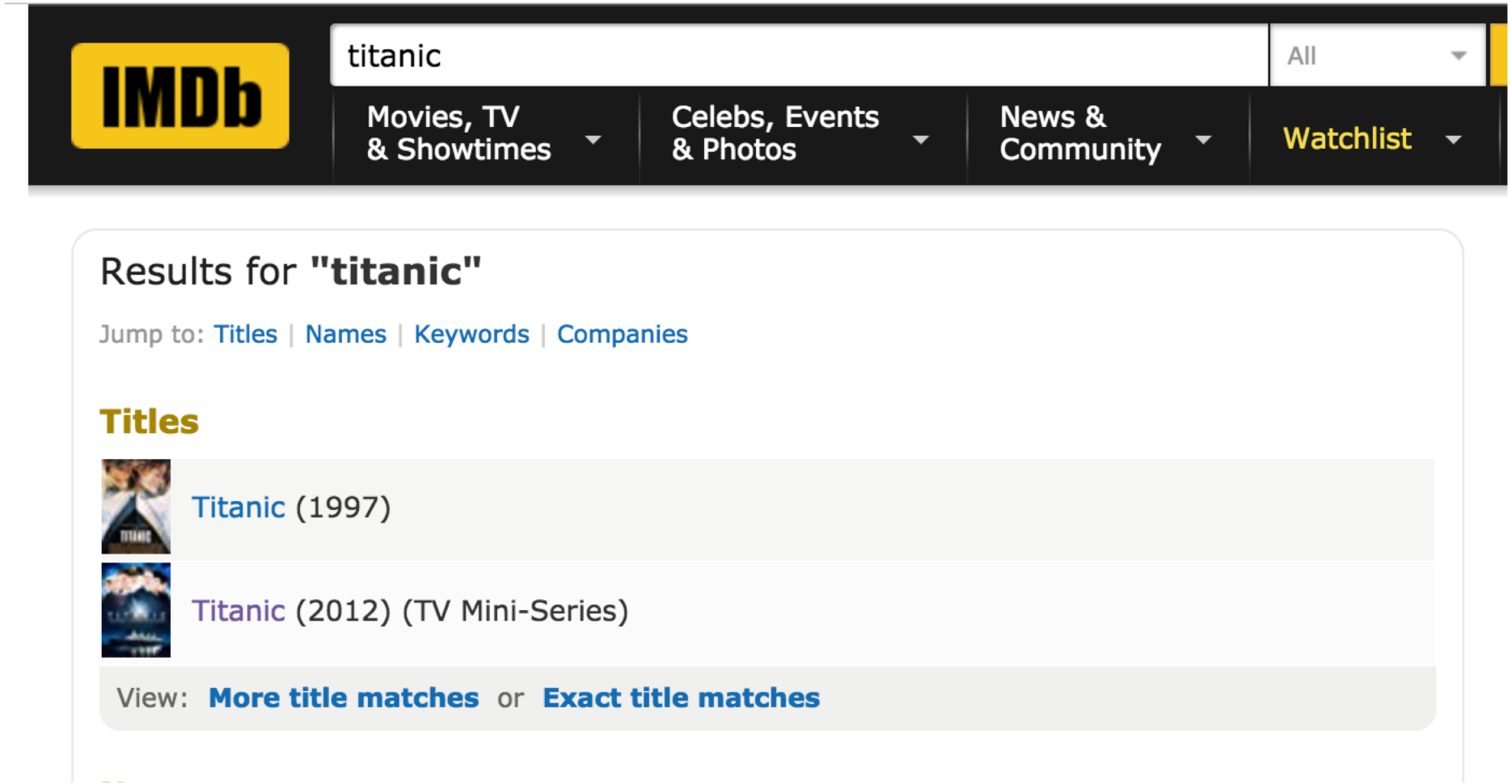
“Movie” table

id	title	rating	year
1	Crazy Rich Asians	PG-13	2018
2	MI: Fallout	G	2018

“Actor” table

id	title	Year_of_birth
1	Henry Golding	1987
2	Constance Wu	1982

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

id	title	rating	year
1	Crazy Rich Asians	PG-13	2018
2	MI: Fallout	G	2018

“Actor” table

id	title	Year_of_birth
1	Henry Golding	1987
2	Constance Wu	1982

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 funcationality of IMDB, but it's essential back-end dev)
- Must use at least one **Database Join**

Before SQLAlchemy, there was **SQL**

"Movies" table

id	title	rating	year
1	Crazy Rich Asians	PG-13	2018
2	MI: Fallout	G	2018

We want to be able to query the database. For example, find an 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
$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
$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();
?>
```

SQLAlchemy is a Python wrapper around SQL

```
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 funcationality 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

id	title	rating	year
1	Crazy Rich Asians	PG-13	2018
2	MI: Fallout	G	2018

"Actor" table

id	title	Year_of_birth
1	Henry Golding	1987
2	Constance Wu	1982

Can we add an Actors column to the Movie table?

"Movie" table

id	title	rating	year	actors
1	Crazy Rich Asians	PG-13	2018	Henry Golding, Constance Wu
2	MI: Fallout	G	2018	Tom Cruise, Alec Baldwin, Superman

"Actor" table

id	title	Year_of_birth
1	Henry Golding	1987
2	Constance Wu	1982

Can we add an actors column to the Movie table AND a movies column to the Actor table?

"Movie" table

id	title	rating	year	actors
1	Crazy Rich Asians	PG-13	2018	Henry Golding, Constance Wu
2	MI: Fallout	G	2018	Tom Cruise, Alec Baldwin, Superman

"Actor" table

id	title	Year_of_birth	Movies
1	Henry Golding	1987	Crazy Rich Asians
2	Constance Wu	1982	Crazy Rich Asians
3	Tom Cruise	1962	MI 1, MI 2, MI 3, MI4, MI5, MI: Fallout

Can we add an actors column to the Movie table AND a movies column to the Actor table?

"Movie" table

id	title	rating	year	actors
1	Crazy Rich Asians	PG-13	2018	1, 2
2	MI: Fallout	G	2018	3

"Actor" table

id	title	Year_of_birth	Movies
1	Henry Golding	1987	1
2	Constance Wu	1982	1
3	Tom Cruise	1962	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Join tables (also called Association tables)

"Movie" table

id	title	rating	year	
1	Crazy Rich Asians	PG-13	2018	
2	MI: Fallout	G	2018	

"Actor" table

id	title	Year_of_birth	
1	Henry Golding	1987	
2	Constance Wu	1982	
3	Tom Cruise	1962	

Movie_actor_join

Movie_id	Actor_id

Join tables (also called Association tables)

"Movie" table

id	title	rating	year	
1	Crazy Rich Asians	PG-13	2018	
2	MI: Fallout	G	2018	

"Actor" table

id	title	Year_of_birth	
1	Henry Golding	1987	
2	Constance Wu	1982	
3	Tom Cruise	1962	

Movie_actor_join

Movie_id	Actor_id
1	1
1	2
2	3
3	3
4	3
5	3

Now quering for actors and movies is easy

"Movie" table

id	title	rating	year	Actors_backref
1	Crazy Rich Asians	PG-13	2018	(automatic)
2	MI: Fallout	G	2018	

Movie_actor_join

Movie_id	Actor_id
1	1
1	2
2	3
3	3
4	3
5	3

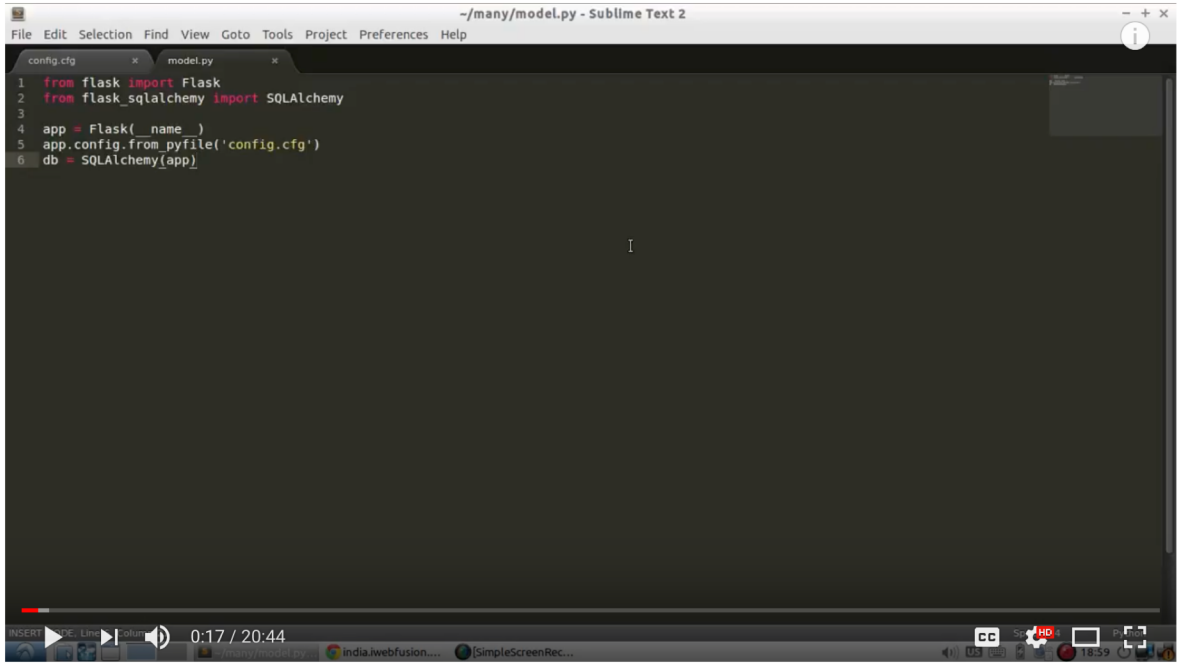
"Actor" table

id	title	Year_of_birth	Movie_backref
1	Henry Golding	1987	(automatic)
2	Constance Wu	1982	
3	Tom Cruise	1962	

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

11,374 views

184 1 SHARE

Pretty Printed
Published on Feb 21, 2016

SUBSCRIBE 19K

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...

SHOW MORE

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 funcationality 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"

3. Report on goal progress.
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.