

Databases, Queries, and Joins

No screens



Prof. Lydia Chilton
COMS 6998
14 September 2018

Say your name



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

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

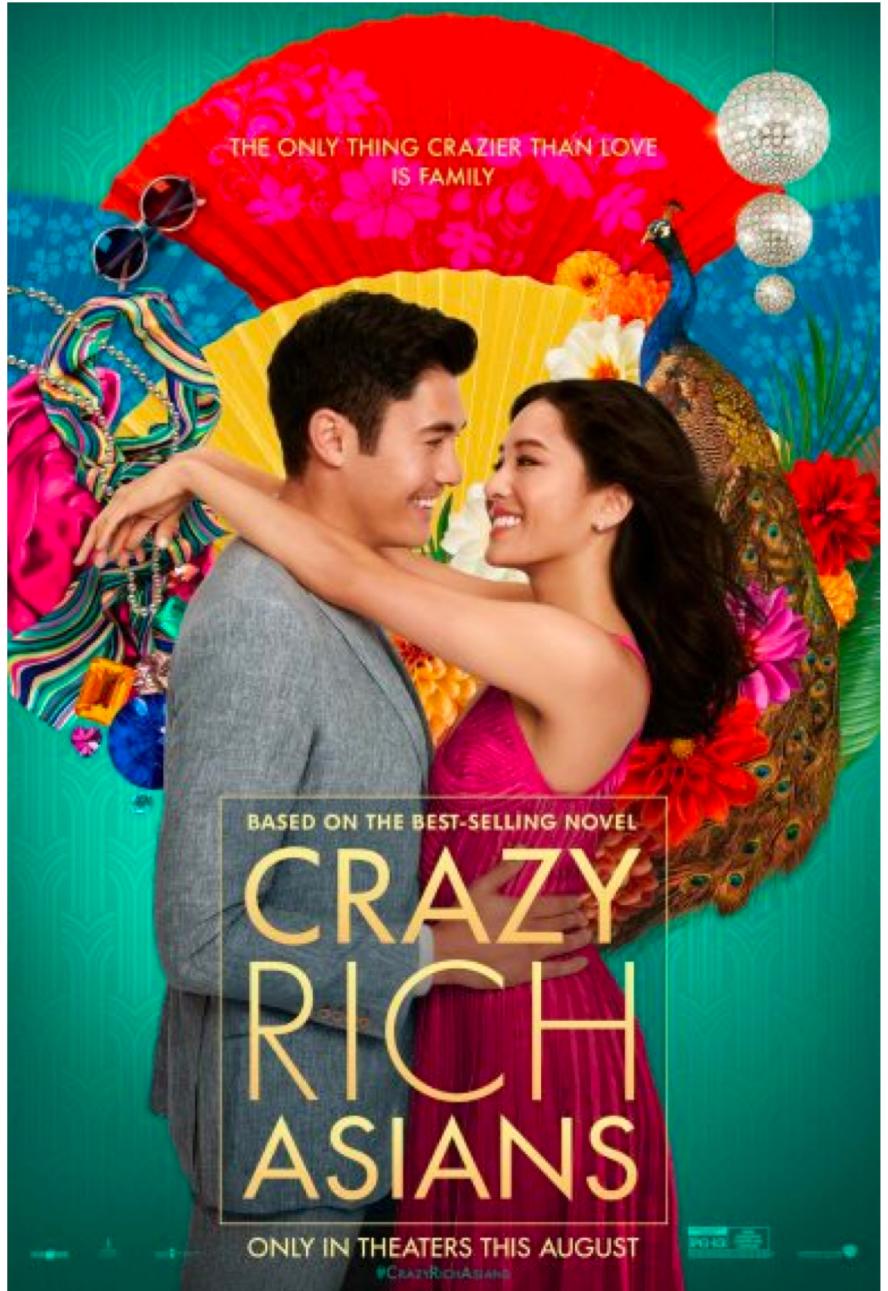
• Pursuing your own project

Rebuilding IMDB.com

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

The IMDb homepage displays several movie trailers and promotional content. At the top, there are three main video thumbnails: "THE NEXT OSCAR CONTENDER?", "The Nun", and "Halloween". Below these, a section titled "Opening This Week" lists movies like "Peppermint", "God Bless the Broken Road", and "Bisbee '17". A large blue button labeled "Get Tickets & Showtimes >" is prominently displayed. The page also features sections for "Now Playing (Box Office)" and "Related News".

A detailed view of the IMDb movie page for "Crazy Rich Asians" (2018). The page includes the movie's title, rating (7.6), and release date (15 August 2018, USA). It features a large image of the movie's trailer, a "Get Showtimes & Tickets" button, and a note about 29 theaters near New York NY. The page also includes a "Videos" section with preview clips and related news articles.



Who is that actor?????

A screenshot of the IMDb website showing the movie "Crazy Rich Asians". The page includes the movie's title, a star rating of 7.6, and a brief plot summary. A green arrow points from the question "Who is that actor?????" towards the IMDb page. The page also shows related news articles and video thumbnails.

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:
 - ?
 - ?
 - ?
 - ?
 - ?
 - ?
 - ?
 - ?
 - ?
 - ?

The top screenshot shows the IMDb homepage. It features a search bar at the top with the placeholder "Find Movies, TV shows, Celebrities and more...". Below the search bar are navigation links for "Movies, TV & Showtimes", "Celebs, Events & Photos", "News & Community", and "Watchlist". There are also "IMDbPro", "Help", and social media links for Facebook, Twitter, and Instagram. A "Sign in with Facebook" button is visible. The main content area includes a banner for "THE NEXT OSCAR CONTENDER?", a section for "5 TIFF Films That Could Be Oscar Contenders", and a "Get Ready for the Toronto International Film Festival" section. To the right, there's a sidebar titled "Opening This Week" listing movies like "The Nun", "Peppermint", and "God Bless the Broken Road". The bottom screenshot shows a detailed movie page for "Crazy Rich Asians" (2018). It includes the movie's title, rating (7.6/10), year (2018), and genre (Comedy, Romance). Below the title are images from the movie, including a couple dancing and a bride. A "Watch now" button is present. The page also includes a "Metascore" from Metacritic (74), "Reviews" (383 user | 130 critic), and "Popularity" (rank 2). A "Videos" section at the bottom shows trailers for the movie.

This is a studio class.
We practice web dev and learn from experience.

Like this:



Not this:



Studio Time

Count off by 3's

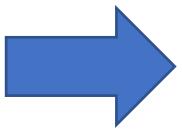
What to discuss during studio

- **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

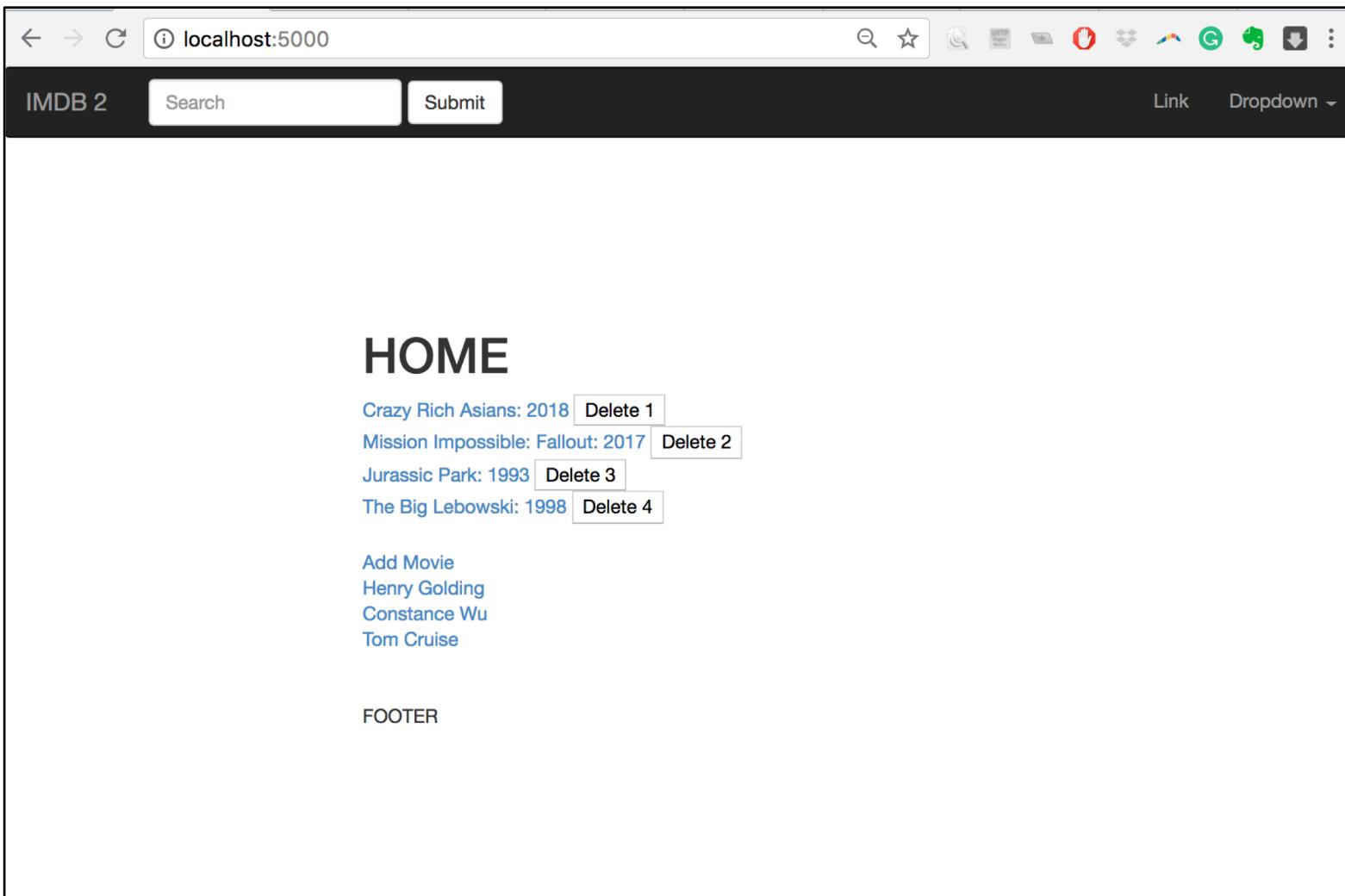
Allow a user to look up the main actors in a movie.

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).

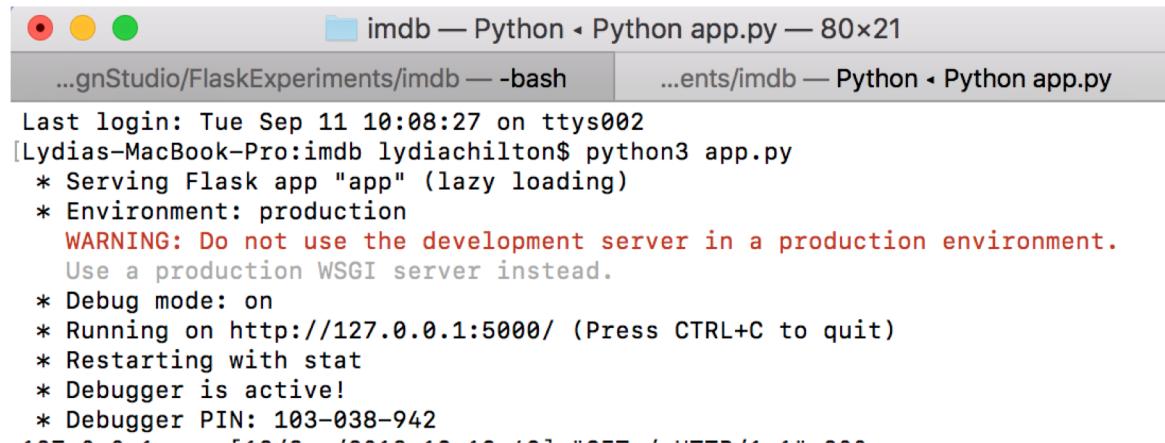
My Homepage



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)

```
19
20  from app import app
21
22
23  if __name__ == "__main__":
24      app.run(debug=True)
```



A screenshot of a Mac OS X terminal window titled "imdb — Python". The window shows the command "python3 app.py" being run in a directory "FlaskExperiments/imdb". The output indicates that the Flask app is running on port 5000 in debug mode, with a warning about not using it in production. It also shows the debugger PIN.

```
Last login: Tue Sep 11 10:08:27 on ttys002
[Lydia's-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.

IMDB 2

Search

Submit

Link

Dropdown ▾

Add Movie

Title:

Year:

Submit

FOOTER

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",
    },
}
```

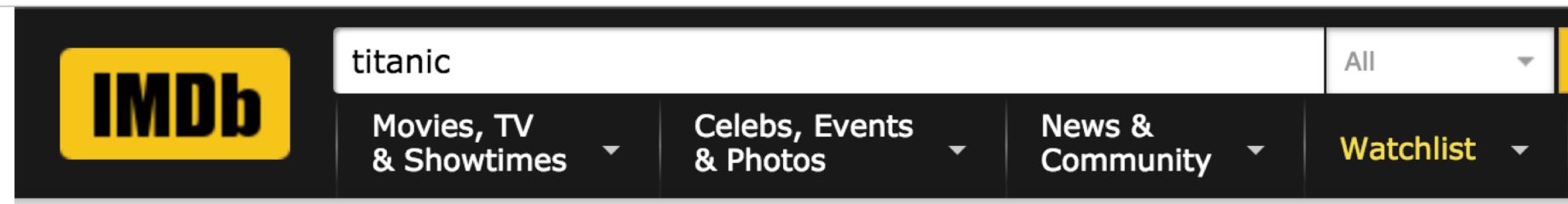
“Movie” table

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???



Results for "titanic"

Jump to: [Titles](#) | [Names](#) | [Keywords](#) | [Companies](#)

Titles



[Titanic \(1997\)](#)



[Titanic \(2012\) \(TV Mini-Series\)](#)

View: [More title matches](#) or [Exact title matches](#)

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 functionality 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 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

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

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 querying 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	

“Actor” table

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

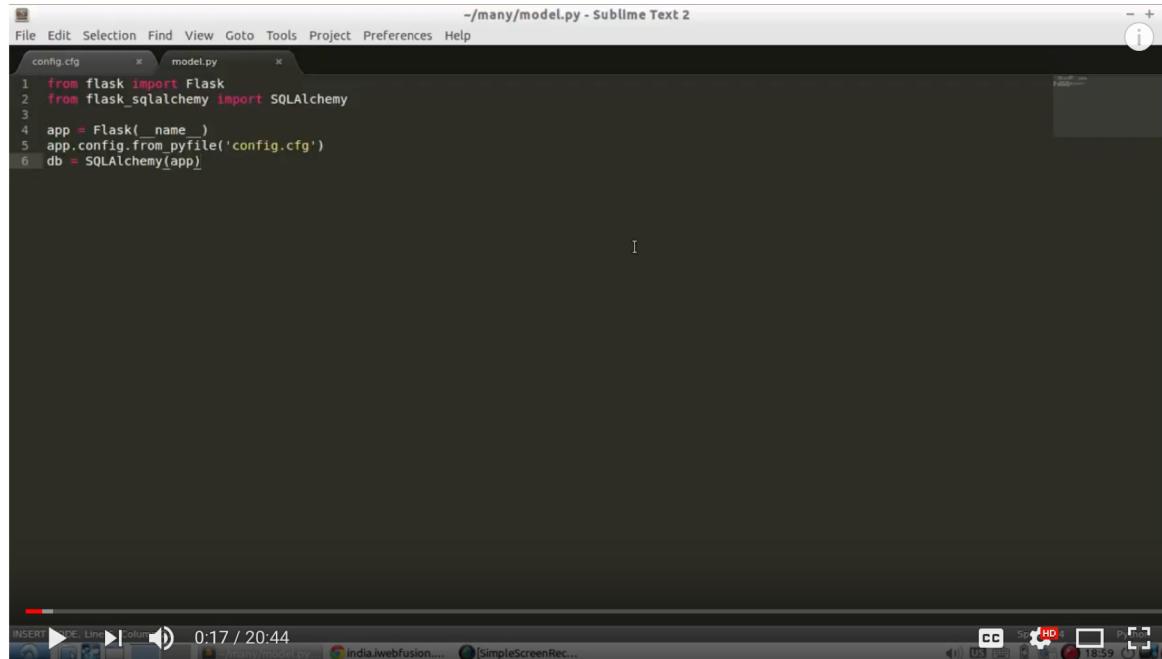
Movie_actor_join

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

```
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 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”

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.