

10. Databases

Logistics

- Demo code for today is already posted!
 - <https://github.com/csc4351-f22/setup-and-demos>
- How are we feeling about the Showcase?
 - No homework this week since I know that stuff is kind of late-breaking
- ProgClub is hosting a HackJam at 3pm today in Student Center East!
 - (but we're covering the same Flask/HTML/CSS stack we've seen in this class)
- Job Readiness content launching on iCollege later today! You can do it and provide feedback for extra credit.

Recap: The Stack So Far

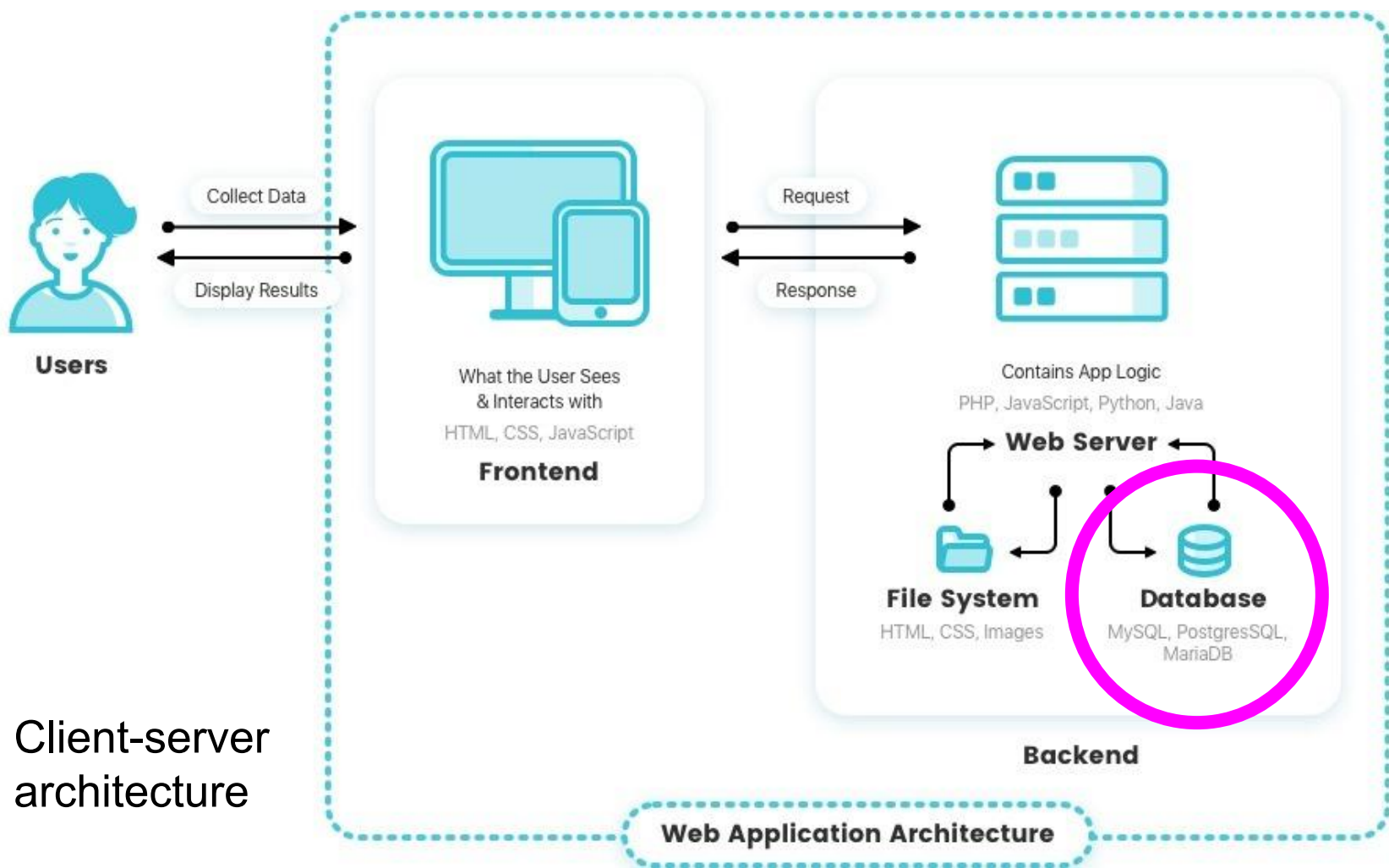
Suppose you want to work in teams to make an exact clone of Twitter (including using Twitter's data) using everything we've covered in the class so far. How does everything we've covered fit in?

Agenda



1. Databases - Theory
2. Databases - Demo

Persisting data



Client-server
architecture

We don't need to save anything in the homework so far...

- Building the layout of our page using templates, HTML, and CSS
- We fetch actual content from external APIs over web requests in our Flask web server code
- We don't actually have to store any of the data we retrieved ourselves!!
- ...but that's all about to change.

How do we go about persisting data in general?

Common Persistence Strategies

- Write stuff out to a file
 - We could use a format like CSV to make sure the file is easily readable
 - We can come up with a special binary format to store our data
 - (Core dumps / error logs use this strategy to persist data)
- We could use a database
 - Involves something like **SQLite**, MySQL, Redis, MongoDB, Cassandra, or PostgreSQL
 - This is common for persisting data that involves tables or objects

Database

... is an organized collection of information (e.g. tables or files) that can be easily stored, managed, and accessed.

Most databases run as a separate service (meaning you must start, install, run, and stop it)

Organization of Databases

- There are two main types: **relational** (SQL) and **non-relational** (NoSQL)
- **Relational databases systems** include SQLite, MySQL, and PostgreSQL
 - They store data in terms of interlinked tables, kind of like Excel
- Interlinked tables make up some crazy math called relational algebra, which allows you to **query** for data using SQL (Structured Query Language)

Organization of Databases

- There are two main types of databases: **relational** (SQL) and **non-relational** (NoSQL)
- **Relational databases systems** include SQLite, MySQL, and PostgreSQL



Organization of Databases

- **Non-relational databases systems** include MongoDB, and Cassandra
 - There's no rule by which these store data, but none use SQL to query
 - MongoDB is **document-based** and stores blobs of JSON
 - Redis and Cassandra are **key-value-based** and are basically like a giant Python dictionary
 - There are also **column-based** and **graph-based**



Seems like a lot... which do I pick ?

It depends...

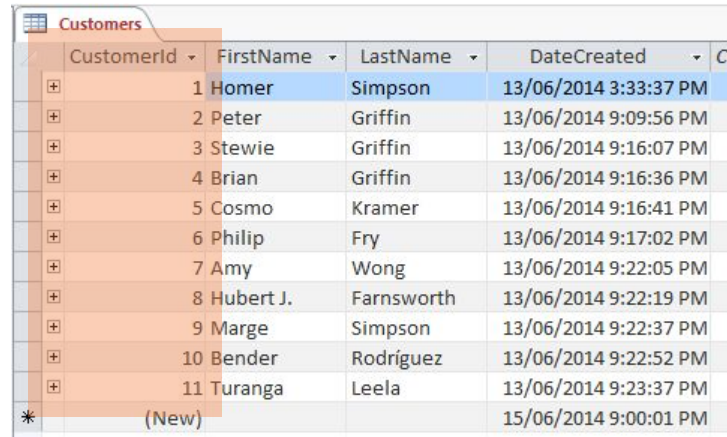
- You can implement all of these using each other
 - For example, you can write a document-based storage service, using a relational database as the underlying store!
- It comes down to a matter of ease-of-use, which means you should pick based on the data you're trying to model
 - Tabular or object-oriented data might be best represented in SQL
 - Object schemas that change constantly or blobs might be best in document stores
- For our project, we're going to use SQLite

Choose your fighter!

- Suppose you want to store an online inventory of products, similar to eBay or Amazon
- ...what kind of database structure makes the most sense? Relational? Non-relational? Specific kinds of either?

Let's first cover some database basics...

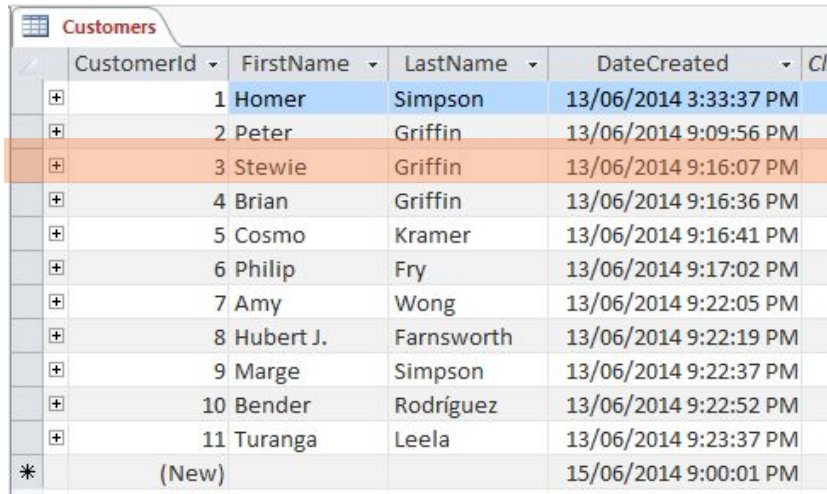
- In a relational database, a table will need to have a **primary key**
 - A primary key is a column in a table that uniquely identifies a row (meaning there is no duplicate value). It should be a number.



CustomerId	FirstName	LastName	DateCreated	Client
1	Homer	Simpson	13/06/2014 3:33:37 PM	
2	Peter	Griffin	13/06/2014 9:09:56 PM	
3	Stewie	Griffin	13/06/2014 9:16:07 PM	
4	Brian	Griffin	13/06/2014 9:16:36 PM	
5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
6	Philip	Fry	13/06/2014 9:17:02 PM	
7	Amy	Wong	13/06/2014 9:22:05 PM	
8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
9	Marge	Simpson	13/06/2014 9:22:37 PM	
10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
11	Turanga	Leela	13/06/2014 9:23:37 PM	
*(New)			15/06/2014 9:00:01 PM	

Let's first cover some database basics...

- A row in a table represents a **record**
 - A record contains every field within a table.

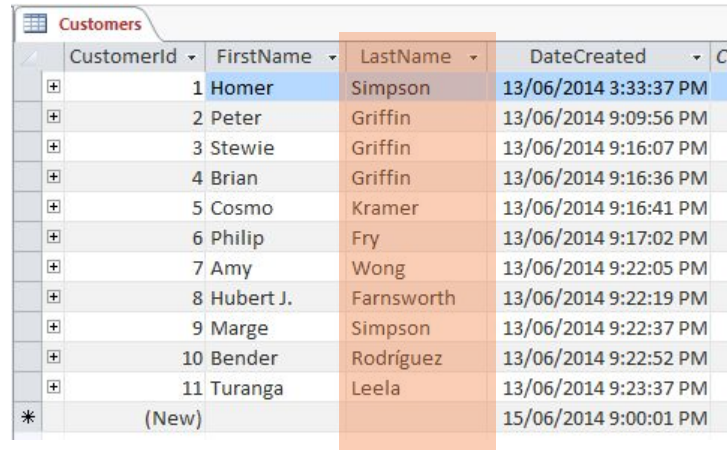


A screenshot of a database application window titled 'Customers'. The window displays a table with the following columns: CustomerId, FirstName, LastName, DateCreated, and a partially visible 'Click' column. The table contains 11 data rows and a final row for a new record. The third row, representing Stewie Griffin, is highlighted in orange. Each data row has a small '+' icon in the first column, and the new record row has a '*' icon.

	CustomerId	FirstName	LastName	DateCreated	Click
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Let's first cover some database basics...

- A column in a table represents **fields**
 - A field contains a single data item within a record.
 - The **key field** is the title of the data contained in the column (ok to just say “column”)



	CustomerId	FirstName	LastName	DateCreated	Cli
+	1	Homer	Simpson	13/06/2014 3:33:37 PM	
+	2	Peter	Griffin	13/06/2014 9:09:56 PM	
+	3	Stewie	Griffin	13/06/2014 9:16:07 PM	
+	4	Brian	Griffin	13/06/2014 9:16:36 PM	
+	5	Cosmo	Kramer	13/06/2014 9:16:41 PM	
+	6	Philip	Fry	13/06/2014 9:17:02 PM	
+	7	Amy	Wong	13/06/2014 9:22:05 PM	
+	8	Hubert J.	Farnsworth	13/06/2014 9:22:19 PM	
+	9	Marge	Simpson	13/06/2014 9:22:37 PM	
+	10	Bender	Rodríguez	13/06/2014 9:22:52 PM	
+	11	Turanga	Leela	13/06/2014 9:23:37 PM	
*	(New)			15/06/2014 9:00:01 PM	

Case Study: Database Schemas

- It can be hard to know what tables you need to create (and what columns those tables need to have) if you're new to interacting with databases!
- So let's think for a second – how could you use a database to implement **user login**? What tables would you need? What columns would the table(s) have? What would a row signify?

Case Study: Database Schemas

- It can be hard to know what tables you need to create (and what columns those tables need to have) if you're new to interacting with databases!
- So let's think for a second – how could you use a database to implement **user login**? What tables would you need? What columns would the table(s) have? What would a row signify?
- What if users were logging into a news site where they could leave comments on each article?

Database Schemas

- **Rule of thumb:** each discrete “object” gets its own table (and each row is one instance of the object)
 - So we implement login with a table of **users**, where each row contains one user’s username and (hopefully hashed) password.
- Each trait of that object is a column in its table
 - So in a table of comments, the columns would be who commented, which article they commented on, etc

SQL: The basics!

The following query syntax will get you through 80% of situations as a software engineer:

- `SELECT * FROM <table_name> WHERE <column name> = <value>`
 - e.g. `SELECT * FROM grades WHERE student_name = "John"`
- We can make this fancier in the following ways:
 - Specify columns to return: `SELECT final_grade, student_id FROM grades WHERE...`
 - Join tables (pull data from two tables at once and cross-reference!): `SELECT final_grade, is_in_programming_club FROM grades JOIN clubs ON grades.student_id=clubs.student_id`

Introducing SQLite (pronounce it however you want)

- It's a **relational** database management system
 - This means it stores data in tables with columns of fields and rows of objects
- Like its name suggests, it can be queried via SQL
 - (Because it's a relational database)
- It doesn't use any external services or database management systems – SQLite just creates a .db file on the host machine (in this case, in the folder where your app is running)

What if we want to publish our app?

Any drawbacks to what we've discussed?

What if we want to publish our app?

- Later, we'll switch over to hosting apps on Heroku (I think? It's a little in flux right now)
- SQLite is best for small amounts of data, and most deployment solutions will also provide more “mature” database management systems
- A lot of times, the database(s) will be hosted on a different machine than the web server!

Break

Introducing SQLAlchemy

- To avoid dealing with custom protocols, writing SQL, and converting Python maps into actual objects, we're going to use a tool called SQLAlchemy
- SQLAlchemy is one of many **ORMs (object relational mappers)**, or tools that translate objects from one language to another
 - In this case, from Python to tabular relational data and vice versa!
- With SQLAlchemy, we can directly write Python object code and have it do the packing AND unpacking.

But first we need a table from a database to query

- To get SQLAlchemy to start persisting and querying for data, we need to define a **model** (which translates to a table in SQLite)
 - You can do this by writing a class that extends `db.Model`
- The model can have different **columns** as its properties
 - These are just instance variables declared with `db.Column`
 - Columns can have different types, just like instance variables
- Since a model is just a Python class, it can also have methods!
 - SQLAlchemy ignores these for the most part, but you can use them in your app

It's demo time! 

SQLite Databases

Until next time... 🙌

- No homework! Go to the Showcase!
- Next class, we will talk about more Flask tricks for making databases useful
- Additional Resources:
 - Flask-SQLAlchemy: <https://flask-sqlalchemy.palletsprojects.com/en/2.x/quickstart/>
 - More sample code:
<https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application>