Databases with SQLite

CS302 2019
Hammond Pearce

https://gfycat.com/hotrigidgrub

Introduction

- Systems are often designed around *data*
- Management of data becomes an issue
- Storage, manipulation, loading, unloading, saving…
- Text files, binary files, a mix…
- JSON, XML, unstructured text...

Databases

- Databases provide mechanisms for managing *structured data*
- Essentially a set of spreadsheets
  - Called “tables”
- Data is organised into **rows** and **columns**
- Each **row** can be thought of as an instance of data
- Each **column** is a field or attribute
- Databases are good for *lots* of data
  - E.g. thousands of products, millions of users, hundreds of millions of posts

Databases

- Internally, databases are structured to manipulate data quickly
- They’re made by better programmers than us
- ACID compliant
  - Atomic - a data manipulation or set of data manipulations cannot be interrupted
  - Consistent - data manipulation can take the database only from valid state to valid state
  - Isolation - data manipulations do not affect one another
  - Durable - once data manipulation is complete, the data is safely saved

- How tricky would it be to get all that with a text file?

Database demo

1. *Create a small database example which has a table of users*

Note: Don't rely on the interface… SQL is typically interacted with via code

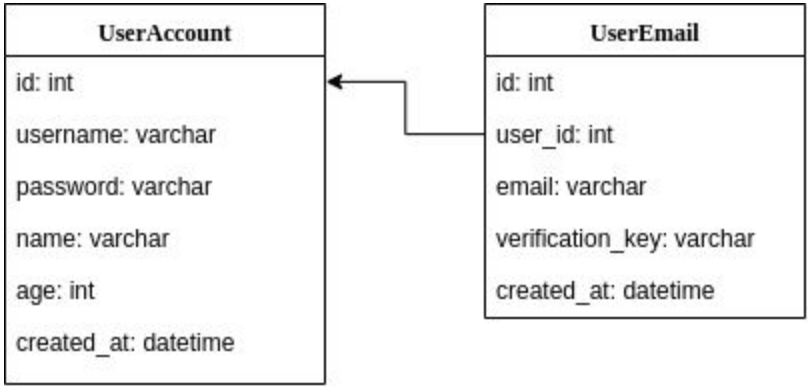
Databases and OO

- Databases are easy when you think in OO
- Each row is an object with attributes
- The table defines the class
  - Each column is attribute with name/type
  - Standard types, from int to string to bool
  - Optional attributes
- Break systems up into objects
- Break objects up onto tables

Relational databases

- Sometimes, objects contain sets of other objects
- E.g. a user may have many email addresses
- *Relational* databases allow us to define relationships between data
  - E.g. a *user_accounts* table has usernames, passwords
  - An *emails* table has email addresses
  - A user_id field is provided in both tables
  - The data can thus be linked

Relational Databases



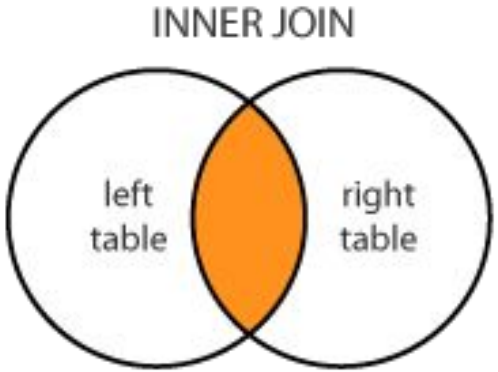
There’s plenty of SQL to go around

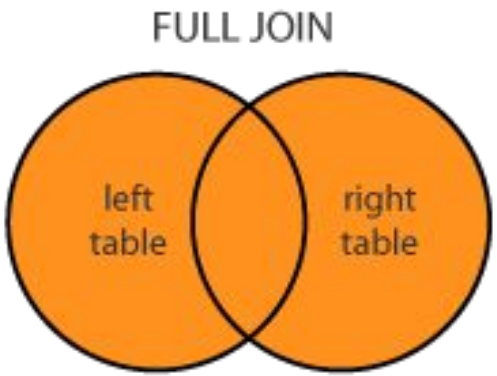
- SQL is a whole language
- Google for the syntax
- CREATE TABLE tblname (col1, col2, col3)
- INSERT INTO tblname (col1, col2) VALUES (val1, val2)
- UPDATE tblname SET (col1 = val1, col2 = val2) WHERE col3 = val3
- DELETE FROM tblname WHERE col3 = val3
- SELECT (...) FROM tblname WHERE col1 = val1
- … joins
  - Inner join
  - Left join
  - Right join

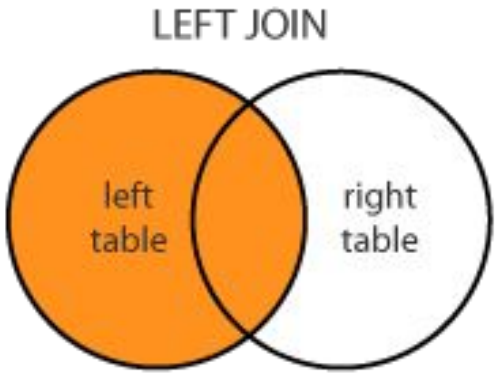
Database Demo - SQLite Browser

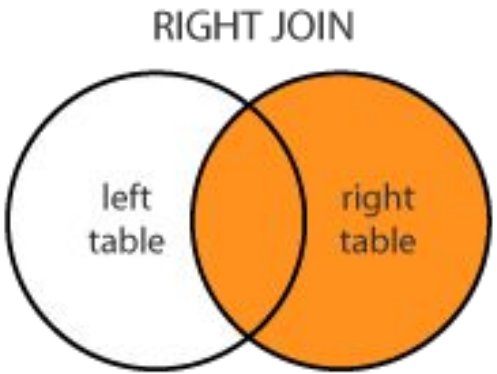
1. *Make a new database*
2. *Create a customers table, add some customers*
3. *Create a emails table, add some emails and link to customers*
4. *Demonstrate select with joins*

SQL Joins (for your interest)









From https://www.dofactory.com/sql/join

And how about from Python?

- It's as easy to SQL in Python as it is to read/write files
- Actually… it's easier!
  - No need to parse files
  - No need to read and write line by line
  - No need to split strings (e.g. for CSV)
- A variety of database *drivers* exist
- We will use the one for SQLite
  - It's in the standard library!

Python and Sqlite

```
import sqlite3
conn = sqlite3.connect("my.db")
#get the cursor (this is what we use to interact)
c = conn.cursor()
#create table
c.execute("""CREATE TABLE users
             (id INTEGER PRIMARY KEY NOT NULL, username TEXT
UNIQUE, password TEXT, age INTEGER)""")
conn.commit()
conn.close()
```

In-class Demo 07.1

From within Python, create a database of users with

ID, USERNAME, PASSWORD, AGE

Add a user, then read it back

What did I do wrong?

- SQL injection occurs when strings are not properly formatted
- Ryan showed you this last Wednesday
- Let’s see an example

In-class Demo 07.2

Check if a username and password match

Use SQL injection to bypass the check

SQL injection is *everywhere*

- Just last week I was asked to solve a database connectivity issue in a real client website
- It is vulnerable to SQL injection
- I informed the client
- They weren't happy, but don't have the budget to fix it
  - Legacy PHP code, very spaghetti, database logic literally everywhere
- It's inexcusable that a developer gave them this code

- So…. how to solve?

In-class Demo 07.3

Fix the code from demo 07.2 so that it is not vulnerable to SQL injection

Conclusions

- SQL makes it much easier to store data compared with other approaches
- Most data is relational, so relational databases are a good fit
- Other types of databases do exist
  - Key/value, Unstructured NoSQL databases
- But SQL is persistent
- Incredibly valuable skill to have
  - Why pivot table and crazy Excel formula when you could just SQL your data?
  - Import from CSV…
- We encourage you all to use SQL within the project