CSE 154

Lecture 21: Introduction to SQL

Administrivia

CP 7 is due Monday - I have office hours after lecture today!

Pokedex is due next Wednesday

Review: Course Outline

✓ HTML: Webpage content

✓ CSS: Webpage presentation

✓ JavaScript: Webpage functionality (client-side)

✓ AJAX: Fetching data from the internet

✓ JSON: JavaScript Object Notation for parsing data nicely

✓ PHP: Server-side code with PHP

SQL: Storing data

Regular Expressions: Validating input

Review: What We've Learned So Far (more specifically)

- How to write **content** for a webpage using `HTML`
- How to add **styles** to a webpage using `CSS` and linking a `CSS` file to an `HTML` file
- How to inspect the `HTML` and `CSS` of web pages in the browser
- How to write behavior for the page and handle user events using JS
- How to fetch and process data from a server using AJAX with JS
- How to write a server-side program and your own API in PHP

A Warmup

(there's no such thing as too much Pokémon...)

A Solution

```js
function filterJSON(pokemonJSON) {
  let pokemon = pokemonJSON["pokemon"];
  let filtered = [];

  // filter out only the Pokemon we want
  for (let i = 0; i < pokemon.length; i++) {
    let data = pokemon[i];
    if (data["weakness"] === "ground" &&
        data["name"].indexOf("a") !== -1) && data["id"] < 150) {
      filtered.push(
        { "name" : data["name"], "type" : data["type"],
          "id" : data["id"], "weakness" : data["weakness"]});
    }
  }

  // Sort (descending by name alphabetically)!
```

JS

Discussion

What are some limitations of filtering out the data in JSON?

- Efficiency: Expensive looping for large datasets, various if/else checks, and extra loops for sorting.
- Long, complex code just to get a subset of the desired data - this code also will need to be manually edited to a good degree when you want a different subset/sorting of data. This also adds more room for programming errors.
- This code was written on the client-side (JS) - could take a relatively long time to finish when the page is also working on other tasks (e.g. user events, other AJAX requests).

Today, you'll learn how to write a clean solution to this problem in SQL

```
function filterJSON(pokemonJSON) {
  let pokemon = pokemonJSON["pokemon"];
  let filtered = [];
  for (let i = 0; i < pokemon.length; i++) {
    let data = pokemon[i];
    if (data["weakness"] === "ground" &&
        data["name"].indexOf("a") !== -1) && data["id"] < 150) {
      filtered.push(
        { "name" : data["name"], "type" : data["type"],
          "id" : data["id"], "weakness" : data["weakness"] });
    }
```

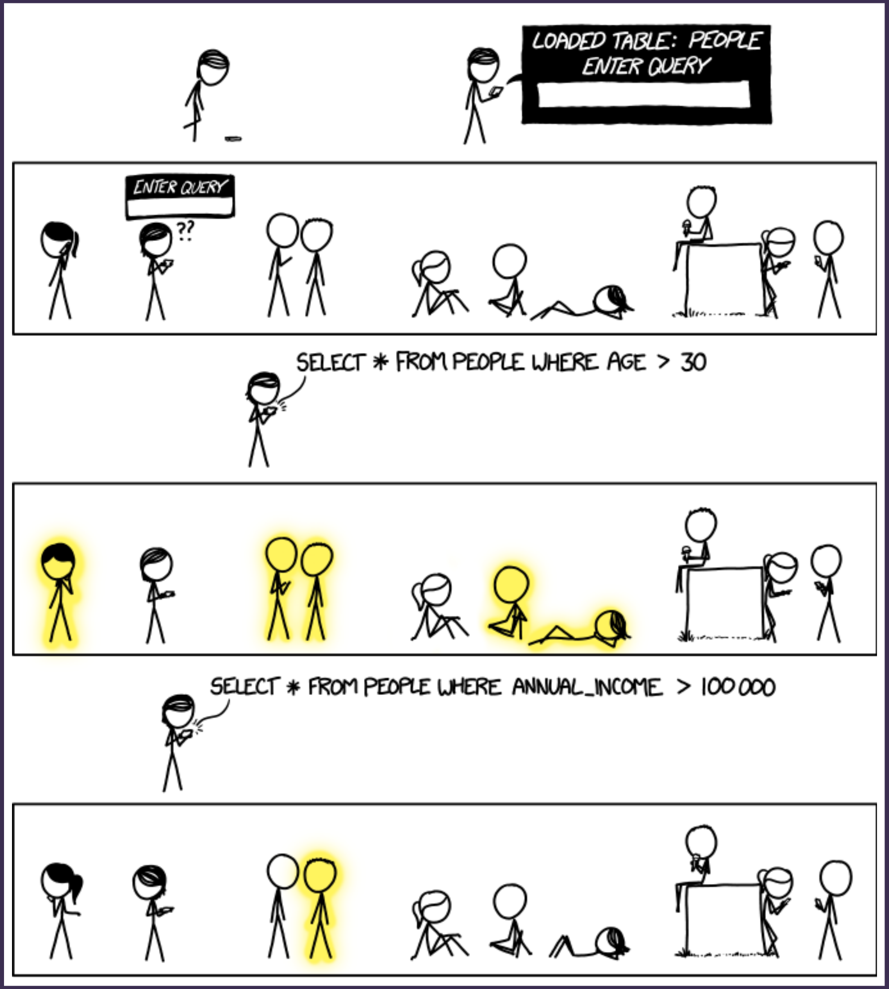
JS

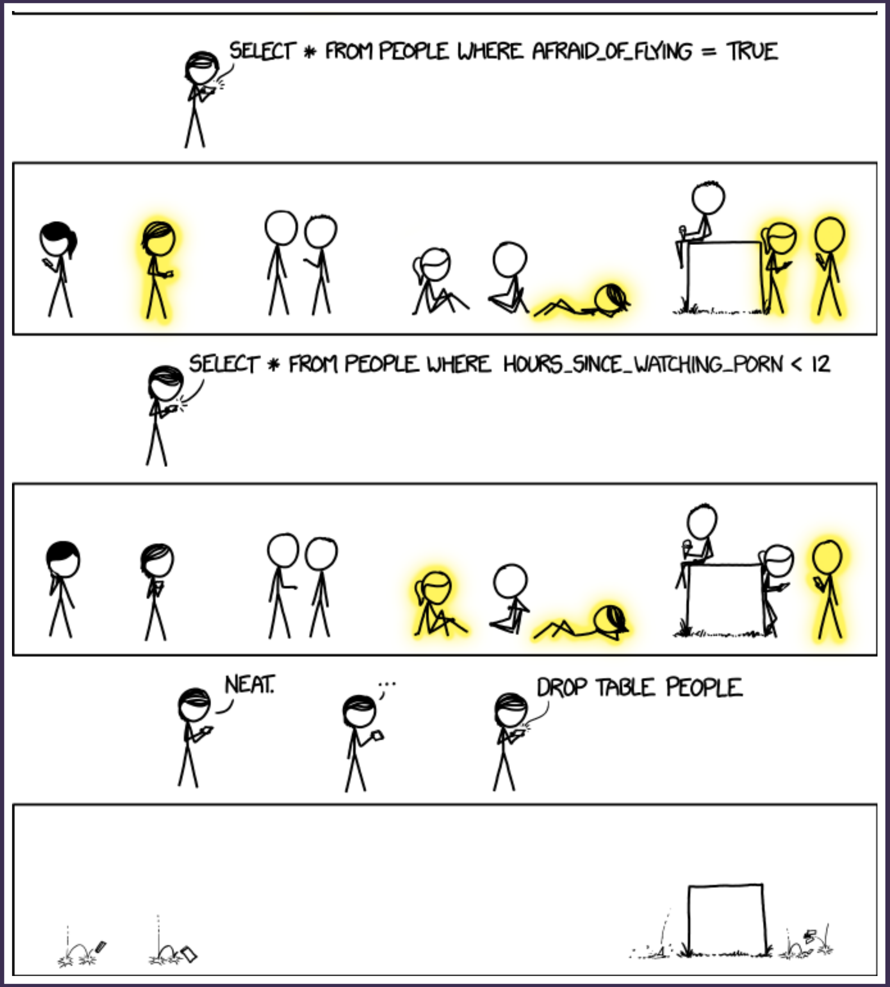
The following code does everything (without JSON/JS) asked in the warmup problem! (Toggle highlighting)

```
SELECT name, id, type, weakness
FROM Pokemon
WHERE id < 150 AND name LIKE %a% AND weakness = 'ground'
ORDER BY type, name DESC;
```

SQL

SQL


LOADED TABLE: PEOPLE
ENTER QUERY
ENTER QUERY
??
SELECT * FROM PEOPLE WHERE AGE > 30
SELECT * FROM PEOPLE WHERE ANNUAL_INCOME > 100 000

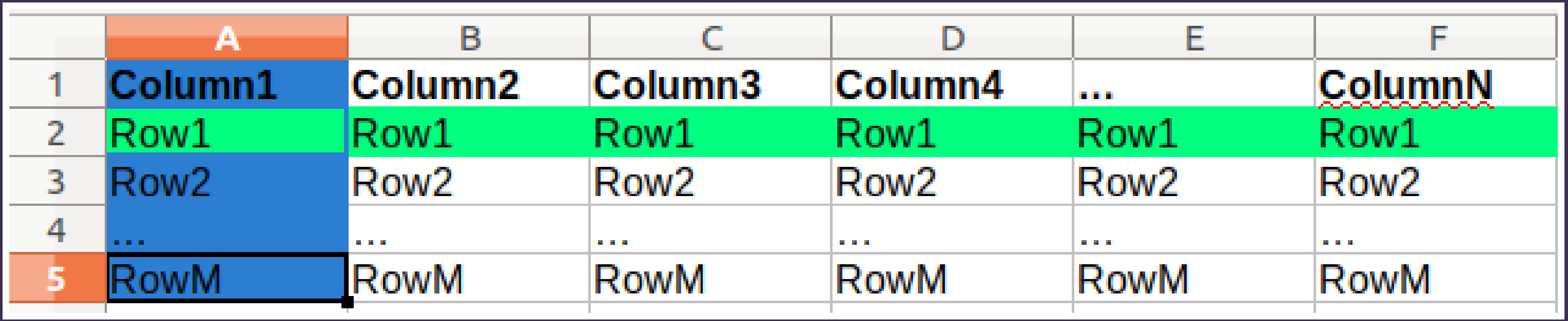

SELECT * FROM PEOPLE WHERE AFRAID_OF_FLYING = TRUE
SELECT * FROM PEOPLE WHERE HOURS_SINCE_WATCHING_PORN < 12
NEAT.
...
DROP TABLE PEOPLE

SQL: a Relational Database Language

Relational Database: A method of structuring data as tables associated by shared attributes

A table row corresponds to a unit of data called a **record**; a column corresponds to an attribute of that record

In Excel speak:



	A	B	C	D	E	F
1	Column1	Column2	Column3	Column4	...	ColumnN
2	Row1	Row1	Row1	Row1	Row1	Row1
3	Row2	Row2	Row2	Row2	Row2	Row2
4	...	...	...	...	...	...
5	RowM	RowM	RowM	RowM	RowM	RowM

In the above image, the cells highlighted blue are the first "column" and the cells highlighted green are of the first "row", or "record"

SQL tables can be visualized just like an Excel sheet, just with different terminology, and more programmatic capabilities.

Guess What? We've Reached the Last Language of 154!

(Insert happy/sad face here depending on your feels)

You can also now appreciate some of this...

Why Learn SQL/Databases in this Class?

Databases give us a great improvement in the way we can build, process, and retrieve large datasets. Most software companies will have a large group dedicated to database management.

Advantages of a database:

- **Powerful**: Can search it, filter data, combine data from multiple sources.
- **Fast**: Can search/filter a database very quickly compared to a file/JSON.
- **Big**: Scale well up to very large data sizes.
- **Safe**: Built-in mechanisms for failure recovery (e.g. **transactions**).
- **Multi-user**: Concurrency feature let many users view/edit data at the same time.
- **Abstract**: Provides layer of abstraction between stored data and app(s) - many database programs understand the same SQL commands.

Database Software

Oracle

Microsoft SQL Server (powerful) and Microsoft Access (simple)

PostgreSQL (powerful/complex free open-source database system)

SQLite (transportable, lightweight free open-source database system)

MySQL (simple free open-source database system)

- Many servers run "LAMP" (Linux, Apache, MySQL, and PHP)
- Wikipedia is run on PHP and MySQL
- We will use MySQL in this course (supported by Cloud9 - no installation necessary!)

Today's Example Database

Games

id	name	platform	release_year	genre	publisher	developer	rating
1	Pokemon Red/Blue	GB	1996	Role-Playing	Nintendo	Nintendo	E
2	Spyro Reignited Trilogy	PS4	2018	Platform	Activision	Toys for Bob	E
3	Universal Paperclips	PC	2017	World Domination	Frank Lantz	Frank Lantz	E
...	...	...	...	...	...	...	...

SQL Basics

```
SELECT name FROM Games WHERE id=23;
```

```
INSERT into Games VALUES
   (514,
   'CSE 154 Pokemon',
   'PC',
   '2017',
   'Card',
   'Melissa Medsker/Whitaker Brand',
   'UW CSE',
   'E');
```

SQL (Games DB)

Structured Query Language (SQL): A language for searching/updating a database.

A standard syntax that is used by all database software (with minor compatibilities)

- Generally case-insensitive

A **declarative** language: describes what data you are seeking, not exactly how to find it.

SQL Statements We'll Cover Today

- SELECT
- DISTINCT
- WHERE
- LIKE
- ORDER BY
- LIMIT

These are the basic "building-blocks" of forming "questions" (queries) in SQL

The SQL `SELECT` Statement

Syntax:

```
SELECT column(s) FROM table;
```

Example:

```
SELECT name, release_year FROM Games;
```

Example output:

name	release_year
Pokemon Red/Blue	1996
Spyro Reignited Trilogy	2018
Universal Paperclips	2017
Super Mario Bros.	1985
...	...

The `SELECT` statement is used to select data from a database and returns the data in a result table containing the row data for column name(s) written after `SELECT` filter.

Table and column names are case-sensitive.

The `DISTINCT` Modifier

Syntax:

```
SELECT DISTINCT column(s) FROM table;
```

The `DISTINCT` eliminates duplicates from the result set.

Example (without `DISTINCT`):

```
SELECT release_year
FROM Games;
```

release_year
1996
2018
2017
1985
1996
2008
...

Example (with `DISTINCT`):

```
SELECT DISTINCT release_year
FROM Games;
```

release_year
1996
2018
2017
1985
2008
...

The SQL `WHERE` Statement

Syntax:

```
SELECT column(s) FROM table WHERE condition(s);
```

Example:

```
SELECT name, release_year FROM Games WHERE genre = 'puzzle';
```

Example result:

name	release_year
Tetris	1989
Brain Age 2: More Training in Minutes a Day	2005
Pac-Man	1982
...	...

The `WHERE` clause filters out rows based on their columns' data values. In large databases, it's critical to use a `WHERE` clause to reduce the result set in size.

Suggestion: When trying to write a query, think of the `FROM` part first, then the `WHERE` part, and lastly the `SELECT` part.

More about the `WHERE` Clause

Syntax:

```
WHERE column operator value(s)
```

Example:

```
SELECT name, release_year
FROM Games
WHERE release_year < 1990;
```

Example result:

name	release_year
Super Mario Bros.	1985
Tetris	1989
Duck Hunt	1984
...	...

The `WHERE` portion of a `SELECT` statement can use the following properties:

- =, >, >=, < <=
- <> or != (not equal)
- min `AND` max
- `LIKE` pattern
- `IN` (value, value, ..., value)

Multiple `WHERE` Clauses: AND, OR

Example:

```
SELECT name, release_year FROM Games
WHERE release_year < 1990 AND genre='puzzle';
```

Example result:

name	release_year
Tetris	1989
Pac-Man	1982
Dr. Mario	1989
...	...

Multiple `WHERE` conditions can be combined using `AND` or `OR`.

Approximate Matches with `LIKE`

Syntax:

```
WHERE column LIKE pattern
```

Example:

```
SELECT name, release_year FROM Games
WHERE name LIKE 'Spyro%'
```

Example results:

name	release_year
Spyro Reignited Trilogy	2018
Spyro the Dragon	1998
Spyro: Year of the Dragon	2000
Spyro 2: Ripto's Rage	1999
...	...

- `LIKE 'text%'` searches for text that starts with a given prefix
- `LIKE '%text'` searches for text that ends with a given suffix
- `LIKE '%text%'` searches for text that contains a given substring

Sorting By a Column: `ORDER BY`

Syntax:

```
SELECT column(s) FROM table
ORDER BY column(s) ASC|DESC;
```

Example (ascending order by default):

```
SELECT name FROM Games
ORDER BY name
```

name
'98 Koshien
007 Racing
007: Quantum of Solace
007: The World is not Enough
...

Example (descending order):

```
SELECT name FROM Games
ORDER BY name DESC
```

name
Zyuden Sentai Kyoryuger: Game de Gaburincho
Zwei
Zumba Fitness: World Party
Zumba Fitness Rush
...

The `ORDER BY` keyword is used to sort the result set in ascending or descending order (ascending if not specified)

Limiting Rows with `LIMIT`

Syntax:

```
LIMIT number
```

Example:

```
SELECT name FROM Games
WHERE genre='puzzle'
ORDER BY name
LIMIT 3;
```

Example result:

name
100 All-Time Favorites
101-in-1 Explosive Megamix
3D Lemmings

`LIMIT` can be used to get the top-N of a given category. It can also be useful as a sanity check to make sure you query doesn't return 100000 rows.

Additional Practice with SQL Queries

SQLZoo has multiple exercises (with built-in databases you don't need to worry about setting up) for practicing `SELECT`, `WHERE`, `ORDER BY`, `LIMIT`, `LIKE`, etc. We recommend you go through these for additional practice!

GalaXQL: Build/destroy/find galaxies with SQL