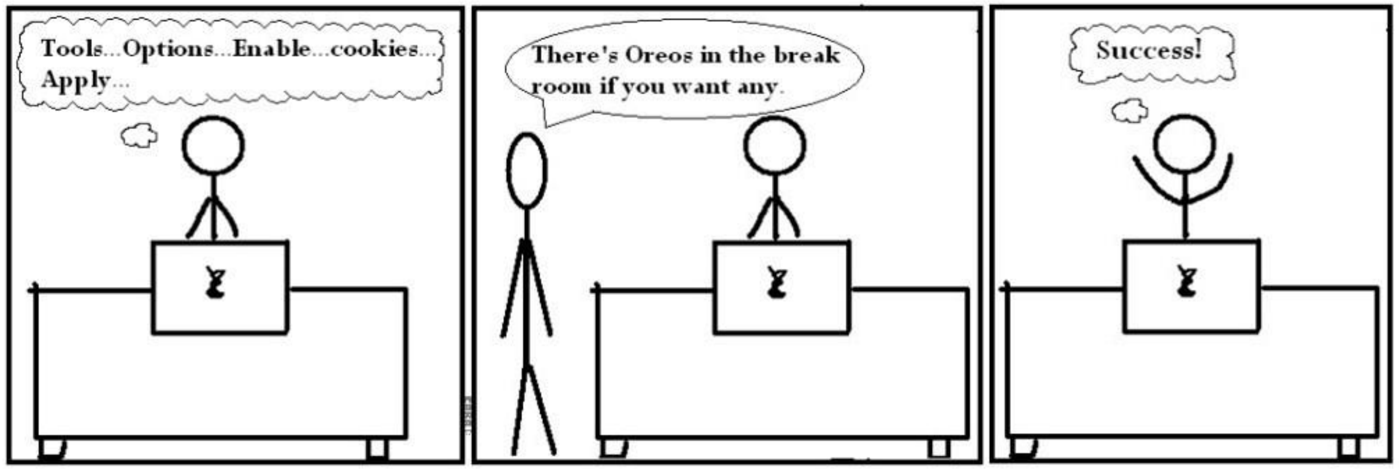
CSC 337


Tools...Options...Enable...cookies...
Apply...
There's Oreos in the break room if you want any.
Success!

LECTURE 20: RELATIONAL DATABASES AND SQL

Relational databases

- relational database: A method of structuring data as tables associated to each other by shared attributes.
- a table row corresponds to a unit of data called a **record**; a column corresponds to an attribute of that record
- relational databases typically use **Structured Query Language** (SQL) to define, manage, and search data

Why use 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
- **big**: scale well up to very large data sizes
- **safe**: built-in mechanisms for failure recovery (e.g. **transactions**)
- **multi-user**: concurrency features let many users view/edit data at same time
- **abstract**: provides layer of abstraction between stored data and app(s)
  - many database programs understand the same SQL commands

Why use SQL?

- Better for relational data
- Still more popular

Strong opinion about this? Let me know!

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

Example csc337simpsons database

id	name	email
123	Bart	bart@fox.com
456	Milhouse	milhouse@fox.com
888	Lisa	lisa@fox.com
404	Ralph	ralph@fox.com

students

id	name
1234	Krabappel
5678	Hoover
9012	Obourn

teachers

id	name	teacher_id
10001	Computer Science 142	1234
10002	Computer Science 143	5678
10003	Computer Science 154	9012
10004	Informatics 100	1234

courses

student_id	course_id	grade
123	10001	B-
123	10002	C
456	10001	B+
888	10002	A+
888	10003	A+
404	10004	D+

grades

- to test queries on this database, use username `csc337homer`, password `d0ughnut`

Example csc337world database

code	name	continent	independence_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...	...	...	...	...	...	...	...

countries (Other columns: region, surface_area, life_expectancy, gnp_old, local_name, government_form, ca pital, code2)

id	name	country_code	district	population
3793	New York	USA	New York	8008278
1	Los Angeles	USA	California	3694820
...	...	...	...	...

cities

country_code	language	official	percentage
AFG	Pashto	T	52.4
NLD	Dutch	T	95.6
...	...	...	...

languages

- to test queries on this database, use username `csc337traveler`, password `packmybags`

Example imdb database

id	first_name	last_name	gender
433259	William	Shatner	M
797926	Britney	Spears	F
831289	Sigourney	Weaver	F
...			

actors

id	name	year	rank
112290	Fight Club	1999	8.5
209658	Meet the Parents	2000	7
210511	Memento	2000	8.7
...			

movies

actor_id	movie_id	role
433259	313398	Capt. James T. Kirk
433259	407323	Sgt. T.J. Hooker
797926	342189	Herself
...		

roles

movie_id	genre
209658	Comedy
313398	Action
313398	Sci-Fi
...	

movies_genres

id	first_name	last_name
24758	David	Fincher
66965	Jay	Roach
72723	William	Shatner
...		

directors

director_id	movie_id
24758	112290
66965	209658
72723	313398
...	

movies_directors

- also available, `imdb_small` with fewer records (for testing queries)

SQL basics

```
SELECT name FROM cities WHERE id = 17;                        SQL
```

```
INSERT INTO countries VALUES ('SLD', 'ENG', 'T', 100.0);      SQL
```

- **Structured Query Language (SQL)**: a language for searching and updating a database
- a standard syntax that is used by all database software (with minor incompatibilities)
    - generally case-insensitive
- a **declarative** language: describes what data you are seeking, not exactly how to find it

The SQL SELECT statement

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

```
SELECT name, code FROM countries;           SQL
```

name	code
China	CHN
United States	IND
Indonesia	USA
Brazil	BRA
Pakistan	PAK
...	...

- the SELECT statement searches a database and returns a set of results
- the column name(s) written after SELECT filter which parts of the rows are returned
- table and column names are case-sensitive

The DISTINCT modifier

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

- eliminates duplicates from the result set

```
SELECT language
FROM languages;          SQL
```

language
Dutch
English
English
Papiamento
Spanish
Spanish
Spanish
...

```
SELECT DISTINCT language
FROM languages;                SQL
```

language
Dutch
English
Papiamento
Spanish
...

The WHERE clause

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

```
SELECT name, population FROM cities WHERE country_code = "FSM";
```

name	population
Weno	22000
Palikir	8600

- `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 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

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

```
SELECT name, gnp FROM countries WHERE gnp > 2000000;    SQL
```

- the `WHERE` portion of a SELECT statement can use the following operators:
  - `=, >, >=, <, <=`
  - `<>` : not equal
  - `BETWEEN` *min* AND *max*
  - `LIKE` *pattern*
  - `IN` (*value*, *value*, ..., *value*)

code	name	gnp
JPN	Japan	3787042.00
DEU	Germany	2133367.00
USA	United States	8510700.00
...	...	...

Multiple WHERE clauses: AND, OR

```
SELECT * FROM cities WHERE code = 'USA' AND population >= 2000000;
```

id	name	country_code	district	population
3793	New York	USA	New York	8008278
3794	Los Angeles	USA	California	3694820
3795	Chicago	USA	Illinois	2896016
…	…	…	…	…

- multiple `WHERE` conditions can be combined using `AND` and `OR`

Approximate matches: LIKE

```
WHERE column LIKE pattern                                    SQL
```

```
SELECT code, name, population FROM countries WHERE name
LIKE 'United%';                                              SQL
```

code	name	population
ARE	United Arab Emirates	2441000
GBR	United Kingdom	59623400
USA	United States	278357000
UMI	United States Minor Outlying Islands	0

- `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

```
ORDER BY column(s)                                          SQL
```

```
SELECT code, name, population FROM countries
WHERE name LIKE 'United%' ORDER BY population;              SQL
```

code	name	population
UMI	United States Minor Outlying Islands	0
ARE	United Arab Emirates	2441000
GBR	United Kingdom	59623400
USA	United States	278357000

- can write ASC or DESC to sort in ascending (default) or descending order:

```
SELECT * FROM countries
ORDER BY population
DESC;                        SQL
```

- can specify multiple orderings in decreasing order of significance:

```
SELECT * FROM countries ORDER BY population DESC, gnp; SQL
```

Limiting rows: LIMIT

```sql
LIMIT number
```
SQL

```sql
SELECT name FROM cities WHERE name LIKE 'K%' LIMIT 5;
```
SQL

name
Kabul
Khulna
Kingston upon Hull
Koudougou
Kafr al-Dawwar

- can be used to get the top-N of a given category (`ORDER BY` and `LIMIT`)
- also useful as a sanity check to make sure your query doesn't return 10^7 rows

Querying databases in Node.js

You will need to install the node package called mysql.

```
npm install mysql
```

Connecting to a database

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: hostname,
  database: databasename,
  user: username,
  password: password,
  debug: "true"
});

con.connect(function(err) {
     if (err) throw err;
     console.log("Connected!");
});
```

Connecting to a Database Example

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "mysql.allisonobourn.com",
  database: "csc337world",
  user: "csc337traveler",
  password: "packmybags",
  debug: "true"
});

con.connect(function(err) {
     if (err) throw err;
     console.log("Connected!");
});
```

Querying a Database

```
var mysql = require('mysql');

var con = mysql.createConnection({
  host: "mysql.allisonobourn.com",
  database: "csc337world",
  user: "csc337traveler",
  password: "packmybags",
  debug: "true"
});

con.connect(function(err) {
      if (err) throw err;
      console.log("Connected!");
      con.query("SELECT * FROM cities WHERE name='london'",
                                          function (err, result, fields) {
             if (err) throw err;
             console.log("Result: " + result[0]["name"]);
      });
});
```

Querying a Database Result

The result object returned by the query is a list of the rows that match the query.

Data for each column can be gotten by accessing the row at the column name.

`result[0]["name"]` from the last slide returns the name of the city in the first returned row.

HTML tables: <table>, <tr>, <td>

A 2D table of rows and columns of data (block element)

```
<table>
  <tr><td>1,1</td><td>1,2 okay</td></tr>
  <tr><td>2,1 real wide</td><td>2,2</td></tr>
</table>
```

HTML

```
1,1             1,2 okay
2,1 real wide   2,2
```

output

- `table` defines the overall table, `tr` each row, and `td` each cell's data
- tables are useful for displaying large row/column data sets
- NOTE: tables are sometimes used by novices for web page layout, but this is not proper semantic HTML and should be avoided

Table headers, captions: <th>, <caption>

```
<table>
  <caption>My important data</caption>
  <tr><th>Column 1</th><th>Column 2</th></tr>
  <tr><td>1,1</td><td>1,2 okay</td></tr>
  <tr><td>2,1 real wide</td><td>2,2</td></tr>
</table>
```
HTML

My important data

Column 1	**Column 2**
1,1	1,2 okay
2,1 real wide	2,2

output

- `th` cells in a row are considered headers; by default, they appear bold
- a `caption` at the start of the table labels its meaning

Styling tables

```
table { border: 2px solid black; caption-side: bottom; }
tr { font-style: italic; }
td { background-color: yellow; text-align: center; width: 30%; }
```

Column 1	**Column 2**
1,1	1,2 okay
2,1 real wide	2,2

My important data

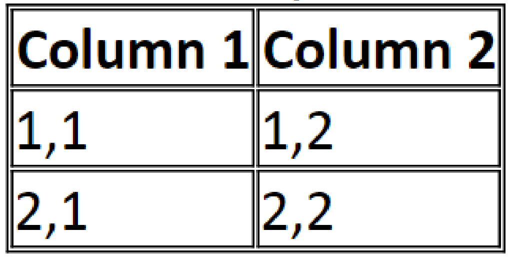
`output`

- all standard CSS styles can be applied to a table, row, or cell
- table specific CSS properties:
  - `border-collapse`, `border-spacing`, `caption-side`, `empty-cells`, `table-layout`

The border-collapse property

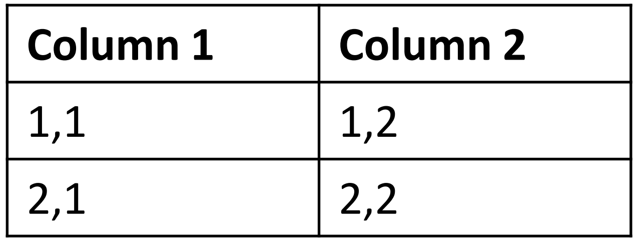
```css
table, td, th { border: 2px solid black; }
table { border-collapse: collapse; }                        CSS
```

Without `border-collapse`



Column 1	Column 2
1,1	1,2
2,1	2,2

With border-collapse



Column 1	Column 2
1,1	1,2
2,1	2,2

- by default, the overall table has a separate border from each cell inside
- the `border-collapse` property merges these borders into one

The rowspan and colspan attributes

```html
<table>
  <tr><th>Column 1</th><th>Column 2</th><th>Column 3</th></tr>
  <tr><td colspan="2">1,1-1,2</td>
    <td rowspan="3">1,3-3,3</td></tr>
  <tr><td>2,1</td><td>2,2</td></tr>
  <tr><td>3,1</td><td>3,2</td></tr>
</table>
```
HTML

Column 1	Column 2	Column 3
1,1-1,2		
2,1	2,2	1,3-3,3
3,1	3,2	

HTML

- `colspan` makes a cell occupy multiple columns; `rowspan` multiple rows
- `text-align` and `vertical-align` control where the text appears within a cell

Column styles: <col>, <colgroup>

```
<table>
  <col class="urgent" />
  <colgroup class="highlight" span="2"></colgroup>

  <tr><th>Column 1</th><th>Column 2</th><th>Column 3</th></tr>
  <tr><td>1,1</td><td>1,2</td><td>1,3</td></tr>
  <tr><td>2,1</td><td>2,2</td><td>2,3</td></tr>
</table>
```
HTML

Column 1	**Column 2**	**Column 3**
1,1	1,2	1,3
2,1	2,2	2,3

output

- `col` tag can be used to define styles that apply to an entire column (self-closing)
- `colgroup` tag applies a style to a group of columns (NOT self-closing

Don't use tables for layout!

- (borderless) tables appear to be an easy way to achieve grid-like page layouts
  - many "newbie" web pages do this (including many UW CSE web pages...)
- but, a `table` has semantics; it should be used only to represent an actual table of data
- instead of tables, use `div`s, widths/margins, floats, etc. to perform layout

- tables should not be used for layout!

- tables should not be used for layout!!

- TABLES SHOULD NOT BE USED FOR LAYOUT!!!

- TABLES SHOULD NOT BE USED FOR LAYOUT!!!!

Designing a query

- Figure out the proper SQL queries in the following way:
  - Which table(s) contain the critical data? (`FROM`)
  - Which columns do I need in the result set? (`SELECT`)
  - How are tables connected (`JOIN`) and values filtered (`WHERE`)?
- Test on a small data set (`imdb_small`).
- Confirm on the real data set (`imdb`).
- Try out the queries first in the MySQL console.
- Write the Node.js code to run those same queries.
  - Make sure to check for SQL errors at every step!!