The Semester so Far

SQL and RDBMS

Select

Every select results in a "new" table

```
SELECT selection_list  # Define what the columns in the relation will be

FROM table_list        # fill in the columns from the listed tables
                       # Does cross product if there are multiple tables

[INNER, RIGHT [OUTER], LEFT [OUTER], FULL [OUTER], NATURAL] JOIN (table [on
constraints])*         # default is to inner join
                       # natural will look for identical column names or
                       # "foreign key" relationships

WHERE constraint+      # Select the rows in the temp table after FROM completes
                       # such that the rows match the given constraint

GROUP BY columns       # groups the remaining rows by the given columns
    HAVING group constraints      # select the grouped rows by the constraint

ORDER BY sorting_cols  # Order the remaining rows by the given columns

LIMIT count;           # Limit on results
```

Joins

```
gtowell=# \d ca
                Table "public.ca"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 one    | integer |           |          |

gtowell=# \d cb
                Table "public.cb"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 two    | integer |           |          |

gtowell=# select * from ca;
 one
-----
   1
   2
   3
   4
   5
(5 rows)

Time: 0.351 ms
gtowell=# select * from cb;
 two
-----
   3
   4
   5
   6
(4 rows)

Time: 0.431 ms
```

For the following Queries answer:

- Legal? (if not fix?)
- What kind of join?
- What are the resulting columns?
- How many rows?
- Contents?
- Rewrite to get same result?

```
select * from ca join cb on one=two;
select * from ca right join cb on one=two;
select * from ca left join cb on one=two;
select * from ca full outer join cb on one=two;
select * from ca cross join cb on one=two;
```

Normalization -- Why?

- Data appears once
  - less storage space
  - easier to update / delete.
- All data is directly queryable

- Minimizes null values by using normalization.
- Results in a more compact database (due to less data redundancy/zero).
- Minimize/avoid data modification problems.
- It simplifies queries???
- The database structure is clearer and easier to understand ???
- The database can be expanded without affecting existing data.
- Finding, sorting, and indexing can be faster because tables are small (fewer columns) so more rows can be accommodated in a single "data page"

Suppose a Database

An Array!

users				
name	company	company_address	url1	url2
Joe	ABC	1 Work Lane	abc.com	xyz.com
Jill	XYZ	1 Job Street	abc.com	xyz.com

```
create table users1 (
        name varchar(20) NOT NULL,
        company varchar(20) not null,
        company_address varchar(20) not null,
        urls varchar[] not null
);

insert into users1 values ('Joe', 'ABC', '1
work Lane', array['abc.com', 'xyz.com']);

insert into users1 (name, company,
company_address, urls) values ('Jill',
'XYZ', '1 Job Street', array['abc.com',
'xyz.com']);
```

```
create table users1 (
        name varchar(20) NOT NULL,
        company varchar(20) not null,
        company_address varchar(20) not null,
        url1 varchar(20) not null,
        url2 varchar(20) not null
);

insert into users1 (name, company,
company_address, url1, url2) values ('Joe',
'ABC', '1 work Lane', 'abc.com', 'xyz.com');

insert into users1 (name, company,
company_address, url1, url2) values ('Jill',
'XYZ', '1 Job Street', 'abc.com', 'xyz.com');
```

DataBase Normalization

- **First Normal Form — There are no multi-valued attributes**
  - Eliminate repeating groups in individual tables.
    - e.g. the url1 & url2 fields
  - Create a separate row in table for each set of related data.
  - Identify each set of related data with a key.
    - So make a “userID” field to indicate that Joe in rows 1 and 2 is the same Joe
      - This is NOT a "primary key"

```
create table users1 (
       name varchar(20) NOT NULL,
       company varchar(20) not null,
       company_address varchar(20) not null,
       url varchar(20) not null,
);
insert into users1 (name, company, company_address,
url) values ('Joe', 'ABC', '1 work Lane',
'abc.com');
insert into users1 (name, company, company_address,
url) values ('Jill', 'XYZ', 'i Job Street',
'xyz.com');
insert into users1 (name, company, company_address,
url) values ('Joe', 'ABC', '1 work Lane',
'abc.com');
insert into users1 (name, company, company_address,
url) values ('Jill', 'XYZ', 'i Job Street',
'xyz.com');
```

users				
userId	name	company	company_address	url
1	Joe	ABC	1 Work Lane	abc.com
1	Joe	ABC	1 Work Lane	xyz.com
2	Jill	XYZ	1 Job Street	abc.com
2	Jill	XYZ	1 Job Street	xyz.com

DataBase Normalization

- **Second Normal Form — Non-key fields must be dependent upon the entire key**
  - Create separate tables for sets of values that apply to multiple records.
    - e.g. the URL field for first normal form
  - Relate these tables with a foreign key.
    - reluserid in URLs is a foreign key to userid in users

users			
userId	name	company	company_address
1	Joe	ABC	1 Work Lane
2	Jill	XYZ	1 Job Street

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

2NF SQL

```
\c company

drop table if exists url2;
drop table if exists user2;

create table user2 (
    userid INT GENERATED ALWAYS AS IDENTITY,
    name varchar(20) NOT NULL,
    company varchar(20),
    company_address varchar(20),
    primary key (userid)
);
insert into user2 (name, company, company_address) values ('Joe', 'xyz', '1 work place');
insert into user2 (name, company, company_address) values ('Jill', 'abc', '1 job street');
insert into user2 (name, company, company_address) values ('Abby', 'xyz', '1 work place');
insert into user2 (name, company, company_address) values ('Loe', 'abc', '1 job street');
create table url2 (
    urlid INT GENERATED ALWAYS AS IDENTITY,
    relUserId int,
    url varchar(20),
    FOREIGN KEY(RelUserId) REFERENCES user3(userid),
    primary key(urlid)
);

insert into url2(relUserId, url) values(1, 'abc.com');
insert into url2(relUserId, url) values(2, 'abc.com');
insert into url2(relUserId, url) values(1, 'xyz.com');
insert into url2(relUserId, url) values(2, 'xyz.com');
```

DataBase Normalization

- **Third Normal Form**
  - **Eliminate fields that do not depend on the key.**
    - The company affiliation of Joe and Jill is NOT an attribute of those people and may be shared

users		
userId	name	relCompId
1	Joe	1
2	Jill	2

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

urls		
urlId	relUserId	url
1	1	abc.com
2	1	xyz.com
3	2	abc.com
4	2	xyz.com

```
drop table if exists urls3;
drop table if exists user3;
drop table if exists company3;

create table company3 (
    companyID INT GENERATED ALWAYS AS IDENTITY,
    company varchar(20),
    company_address varchar(20),
    primary key(companyid)
);
insert into company3 (company, company_address) values ('ABC', '1 Work Lane');
insert into company3 (company, company_address) values ('XYZ', '1 Job Street');
create table user3 (
    userid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name varchar(20) NOT NULL,
    RelCompanyId int,
    FOREIGN KEY(RelCompanyId) REFERENCES company3(companyID)
);
insert into user3 (name, RelCompanyId) values ('Joe', 1);
insert into user3 (name, RelCompanyId) values ('Jill', 2);
create table url3 (
    urlid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    relUserId int,
    url varchar(20),
    FOREIGN KEY(RelUserId) REFERENCES user3(userid)
);
```

4NF

- The URLs table has info in it that is not an attribute of the URL. Namely the “reluser” column.
- This feels wrong.
  - Also, we have data that repeats and could be updated separately. (The URL)
- Clean up by replacing the URL table with two:
  - URLs and URL_relations
- **Fourth Normal Form**
  - **In a many-to-many relationship, independent entities can not be stored in the same table.**

```
create table urls4 (
    urlid int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    url varchar(20)
);
insert into urls4 (url) values ('abc.com');
insert into urls4 (url) values ('xyz.com');
create table url_rel4 (
    relationID int GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    RelURLid int,
    RelCompanyID int,
    FOREIGN KEY(RelURLid) REFERENCES urls4(urlid),
    FOREIGN KEY(RelCompanyId) REFERENCES company4(companyID)
);
insert into url_rel4 (RelURLid, relcompanyid) values (1, 1);
insert into url_rel4 (RelURLid, relcompanyid) values (1, 2);
insert into url_rel4 (RelURLid, relcompanyid) values (2, 1);
insert into url_rel4 (RelURLid, relcompanyid) values (2, 2);
```

users		
userId	name	relCompId
1	Joe	1
2	Jill	2

companies		
compId	company	company_address
1	ABC	1 Work Lane
2	XYZ	1 Job Street

urls	
urlId	url
1	abc.com
2	xyz.com

url_relations		
relationId	relatedUrlId	relatedUserId
1	1	1
2	1	2
3	2	1
4	2	2

Short and Long Polling from the server side

- Short -- respond ASAP
  - even if you have nothing to say
- Long -- respond when you have something to say

Simulate waiting for new information

Kill connection if too long (2.5 sec)
Long poll null return

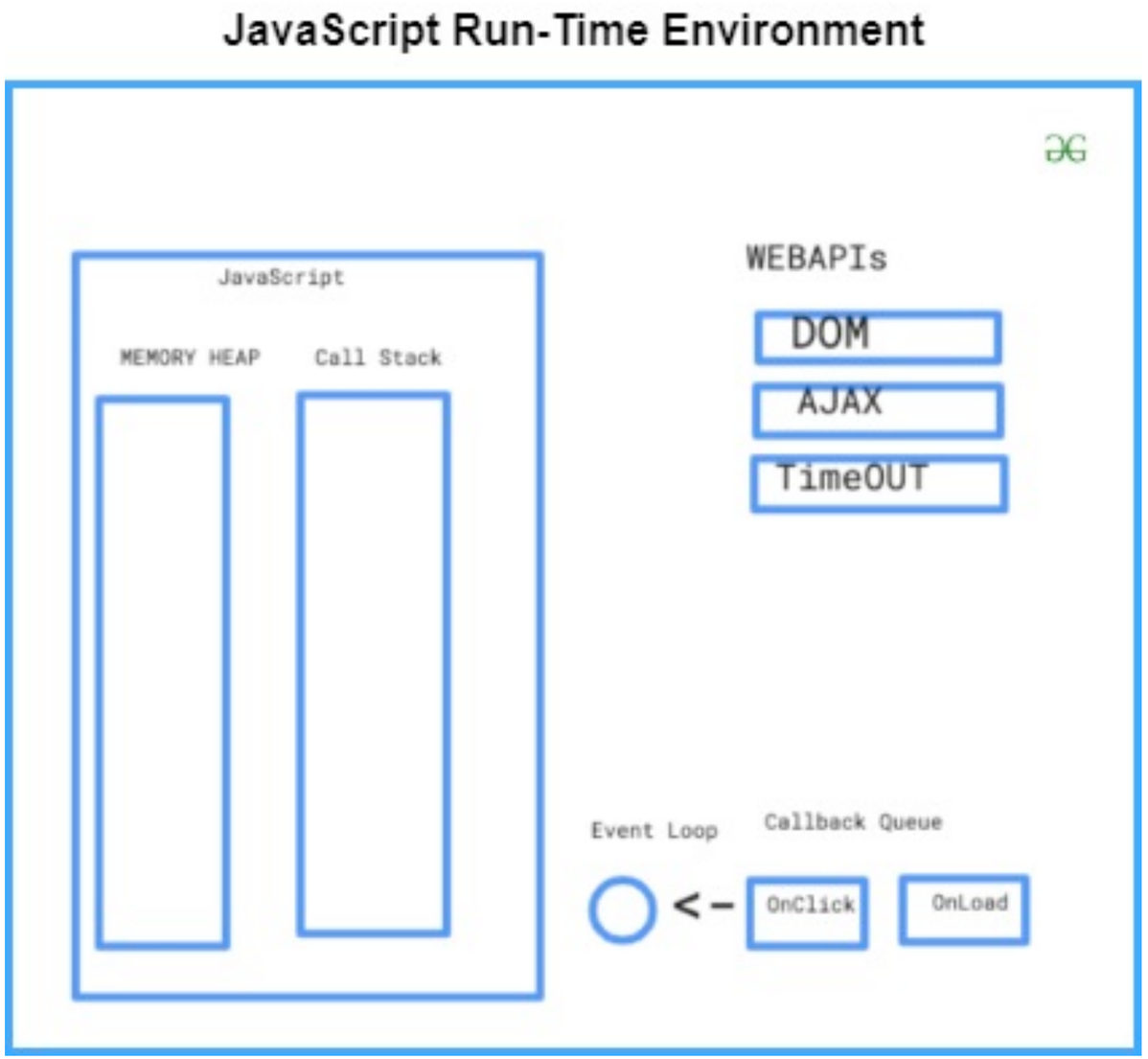
```
const express = require('express');
const app = express();
const url = require('url');
const port = 30001

let counter = 0;
app.use(express.json());
app.use(express.urlencoded());

app.use("/post-test", function (req, res) {
    let tt = Math.random() * 5000;
    console.log(tt)
    setTimeout(() => { writePage(res, req.body) }, tt)
})

// stuff not shown

const server = app.listen(port, function(error){
    if(error) throw error
    console.log("Server created Successfully")
})

server.setTimeout(2500, (socket) => {
    console.log(socket);
    socket.destroy();
  });
```

/home/gtowell/Private/383/NodeIntro24/post_get2.js

Javascript

Javascript is single-threaded and non-blocking

Event Driven



https://www.geeksforgeeks.org/why-javascript-is-a-single-thread-language-that-can-be-non-blocking/

https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop

Single threaded == javascript only ever does one thing at a time

Non-blocking = JS never gets stuck waiting for an event (bad programmers can write a classic sleep loop)

Event-driven = after setup program waits for event and then responds per the program setup.

Overview: What is JavaScript and why do we care?

Web Development

Used to make HTML dynamic and interactive

Event driven and timing

```
<html>
    <body>
        <script>
            let count=0;
            function replaceAfter(timeee) {

document.querySelector("#span1").innerHTML=`Started $
{count}`;
                for(let i=0; i<timeee; i++) {
                    for (let j=0; j<timeee; j++) {
                        for (let k=0; k<timeee; k++) {
                            let z = i*j*k;
                        }
                    }
                    console.log(i);
                }
                count++;

document.querySelector("#span2").innerHTML=`Finished $
{count}`;
            }
        </script>
        <span id="span1" style="font-
size:300%;color:red">Original 1</span>
        <br/>
        <span onclick="replaceAfter(1500);" id="span2"
style="font-size:400%;color:blue">Original 2</span>
    </body>
</html>
```

https://cs.brynmawr.edu/~gtowell/383/js6.html and js7.html

```
<html>
    <body>
        <script>
            let count=0;
            function replaceAfter(timeee) {

document.querySelector("#span1").innerHTML=`Started $
{count}`;
                setTimeout(function() {
                    count++;

document.querySelector("#span2").innerHTML=`Finished $
{count}`;
                }, timeee);
            }
        </script>
        <span id="span1" style="font-
size:300%;color:red">Original 1</span>
        <br/>
        <span onclick="replaceAfter(5000);" id="span2"
style="font-size:400%;color:blue">Original 2</span>
    </body>
</html>
```

- Javascript does not have sleep() because it would be blocking,
  - You can kind of simulate it (as at left)
  - DO NOT!!!

JS and body text

- Evaluation order in html page redux
  - It is not just JS, but everything that is evaluated in order.
    - All DOM is built in textual order
  - So, pages elements are unavailable to JS before they appear
- Conclusion: any JS that modifies a page -- immediately -- should be last

```
<html>
    <body>
        <script>
          document.querySelector("#span1").text="rep 1";
        </script>
        <span id="span1" >Original 1</span>
        <br/>
        <span id="span2" >Original 2</span>
        <script>
          document.querySelector("#span2").innerHTML="rep 2";
        </script>
    </body>
</html>
```

https://cs.brynmawr.edu/~gtowell/383/js3.html

- "Node.js is an open-source, cross-platform runtime environment for JavaScript that executes the code outside of the browser." https://radixweb.com/blog/nodejs-usage-statistics
- "Node.js is designed to build scalable network applications." https://nodejs.org/en/about
- Node.js is system we will use in this class to build and deploy dynamic web sites Geoff

Node: Hello World

- create directory and CD into it
- Determine the "port" you are going to use
  - http://165.106.10.133:30006/index6.html
- npm install express
- make the file
  - hw.js
- start Node with the file
  - UNIX> node hw.js
- In a browser navigate to
  - http://165.106.10.133:30001/helloworld

```
const express = require('express')
const app = express()
const port = 30001

app.use("/helloworld", function (req, res) {
    res.write("<html><body>")
    res.write("Hello World")
    res.end("</body></html>")
})

app.listen(port, function (error) {
    if (error) throw error
    console.log(`Server created on port ${port}`)
})
```

prints to terminal

/home/gtowell/Private/383/NodeIntro24/hw.js