

Servers and Databases

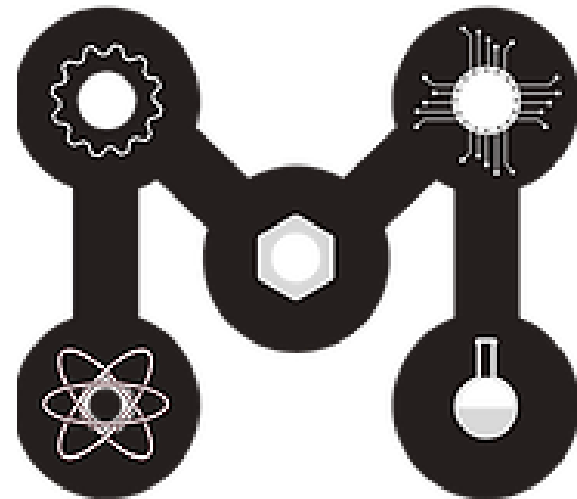
Where all the magic happens

Hosted by:



IEEE

UW Madison




More Workshops This Year!



<https://making.engr.wisc.edu>

← → ↻ ⓘ https://making.engr.wisc.edu

UNIVERSITY of WISCONSIN-MADISON

 **GRAINGER ENGINEERING DESIGN INNOVATION LAB**

HOME GET INVOLVED ^ EQUIPMENT POLICIES v ABOUT US v QUESTIONS & CONTACT INFO

Location

Events

Join the Makerspace email list

Teach Your Class at the Makerspace

Host your Event at the Makerspace

←

IMITLESS

What is a Server?



- Any “Computer” with a way to store data, run processes, and can connect to other computers can be a Server

Connect to Server

- HTTP (Hypertext Transfer Protocol)
 - Way of sending data to servers
 - Just a protocol
- SSH
 - Open a terminal
- FTP
 - Transfer files

Why Protocols

- Way of knowing how data is formatted
- We send 2 bytes of data
 - 4 bits for A, 4 bits for B, 8 bits for C



0100001010100111

Or

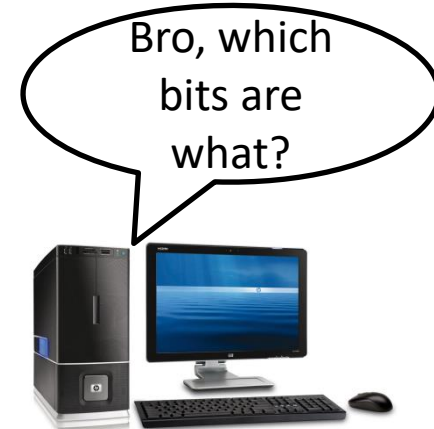
0100001010100111

Or

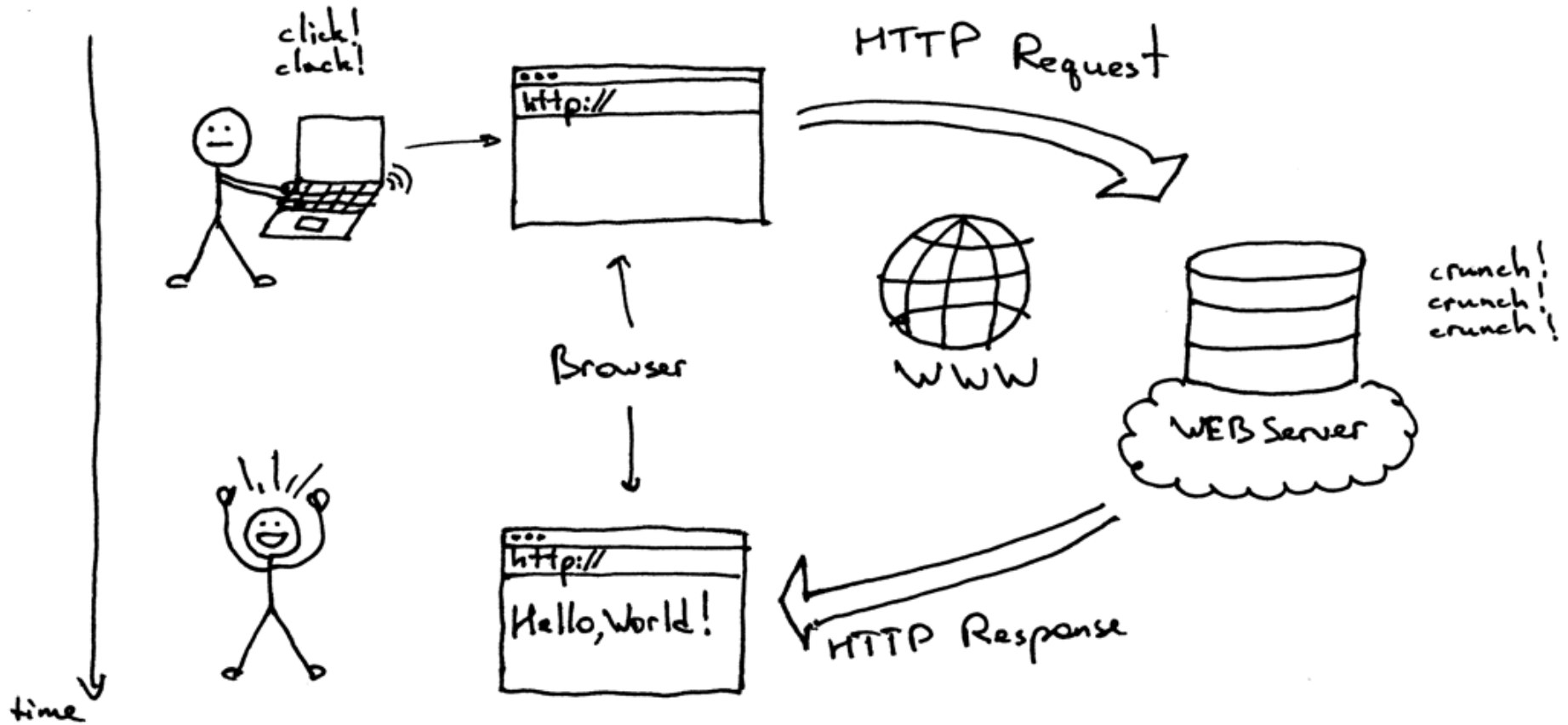
0100001010100111

Or what about

0100001010100111



HTTP Protocol



How do I run a HTTP Server?

- With any language you want
 - Just need to be able to handle HTTP Requests
- Popular Languages and Frameworks
 - Python and Flask
 - Java and Spark
 - JavaScript and NodeJS

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Sending HTTP Request

- GET
 - All information is in URL/Request Line
 - <http://myServer:8888/path/with/info/about/the/get/call>
 - Never send passwords in here
- POST
 - Same as GET but has a **JSON Body** with info
 - Used to send larger amount of data

The Response – Status Code

- Response is just a JSON object
- Status codes to know:
 - **200** – We All Good
 - **4xx's** - Client screwed up
 - **404 Not Found**
 - **5xx's** - The Server screwed up
 - **500 Internal Server Error**

Async Programming

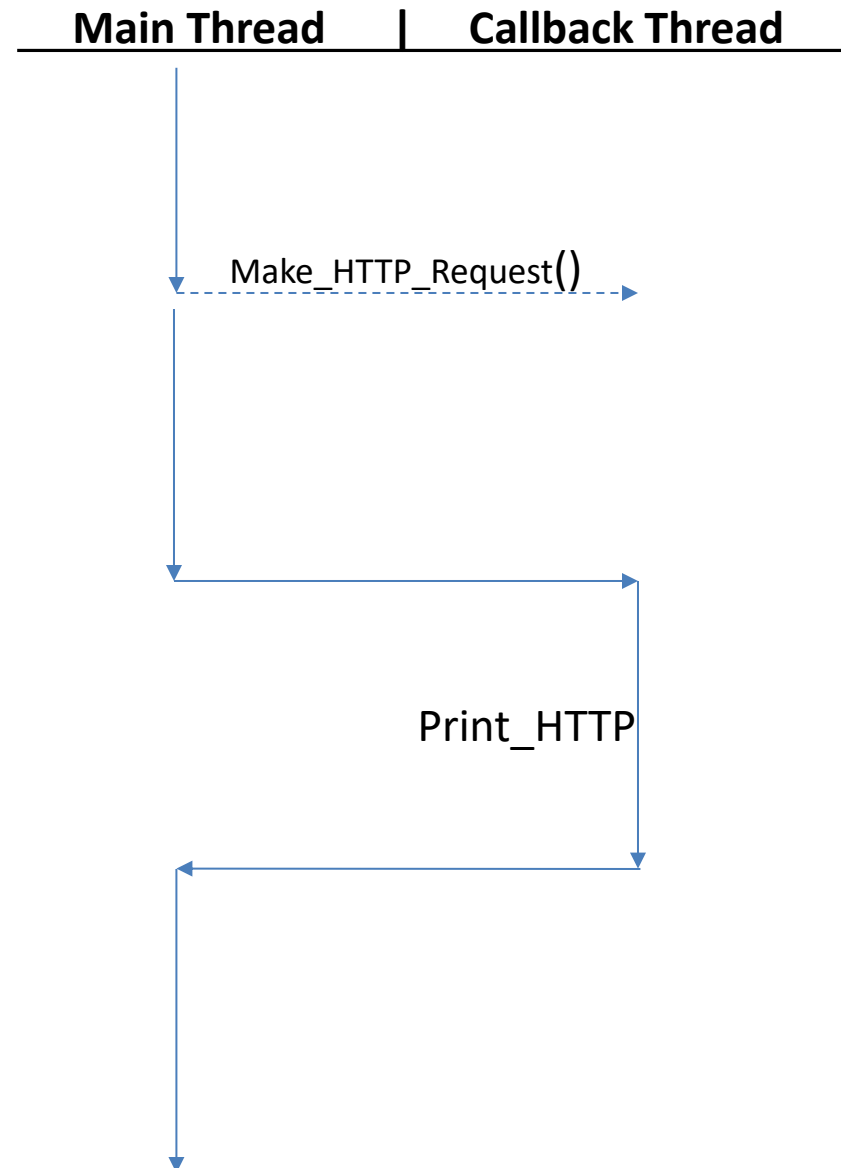
- Code usually is synchronous or “blocking”

Might take long time



```
if (something > 4) {  
    Do_That();  
}  
  
a = b * 3;  
  
Do_Something();  
  
Make_HTTP_Request();  
  
Do_That_Other_Thing();  
...
```

```
Do_Something();  
  
Make_HTTP_Request(Print_HTTP);  
  
Do_That_Other_Thing();  
  
...  
...  
...  
  
function Print_HTTP(response) {  
    // Print code  
}
```



Databases

- Way to save data in persistent memory
- Can host on server
 - Allows to not physically hold data
- Can host on local computer
 - Or in this case the Pi
- No more saving and reading to TXT Files

Every Database is CRUD

- **Create**
 - Add new data to database
- **Read**
 - Get data from database
- **Update**
 - Change data from database
- **Delete**
 - Erase data from database

API vs Database

- API is the **Application** that does the CRUD
- Example
 - <https://api.github.com/users/sjfricke>
 - Fetches JSON data from a Database
 - The backend server code is the API

```
{ "login": "sjfricke",  
  "id": 9061055,  
  "avatar_url": "https://avatars.githubusercontent.com/u/9061055?v=3",  
  "html_url": "https://github.com/sjfricke",  
  "email": "sjfricke@wisc.edu",  
  "updated_at": "2016-10-08T05:48:35Z" }
```



Client

(You on a computer or phone)



Server

(Could be your Raspberry Pi)



Database

(program living on the server)

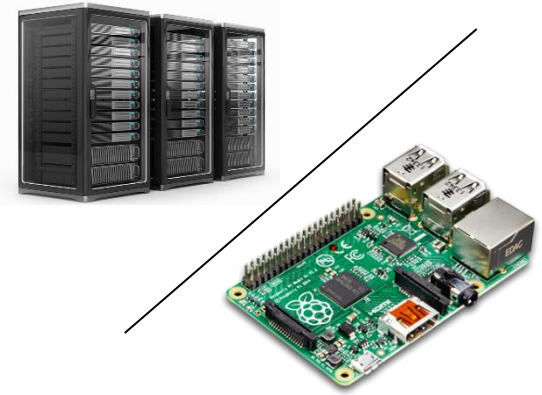
Disclaimer!

- NOT how passwords work
 - Lesson for different day



Client

(You on a computer or phone)



Server

(Could be your Raspberry Pi)



Database

(program living on the server)



Client

(You on a computer or phone)

<http://www.facebook.com/login>

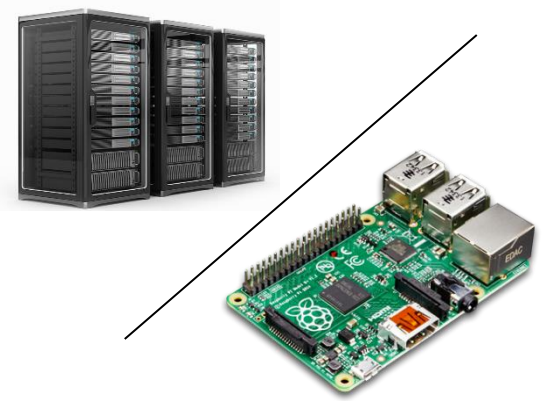
Body:

{

Username: "sjfricke@wisc.edu",

Password: "\$uP3r\$3cRet"

}



Server

(Could be your Raspberry Pi)



Database

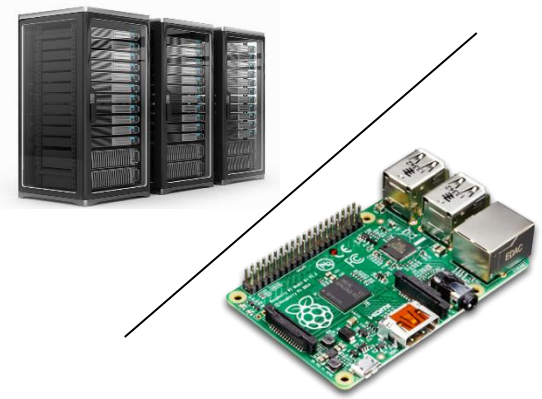
(program living on the server)



Client

(You on a computer or phone)

Makes a **Query** to ask the database
for the information of data where
Username == "sjfricke@wisc.edu"



Server

(Could be your Raspberry Pi)



Database

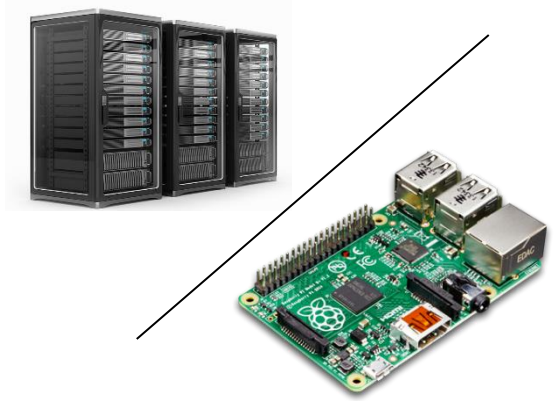
(program living on the server)



Client

(You on a computer or phone)

Returns information including the
password



Server

(Could be your Raspberry Pi)



Database

(program living on the server)



Client

(You on a computer or phone)

The Server logic

```
if (password_sent == password_from_query) {  
    valid_login = true;  
} else {  
    valid_login = false;  
}
```



Server

(Could be your Raspberry Pi)



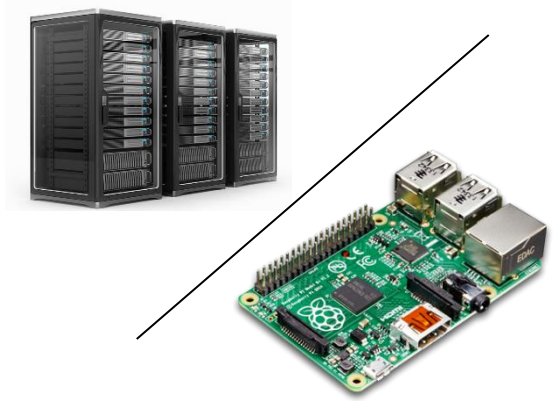
Database

(program living on the server)



Client

(You on a computer or phone)



Server

(Could be your Raspberry Pi)

The Server logic

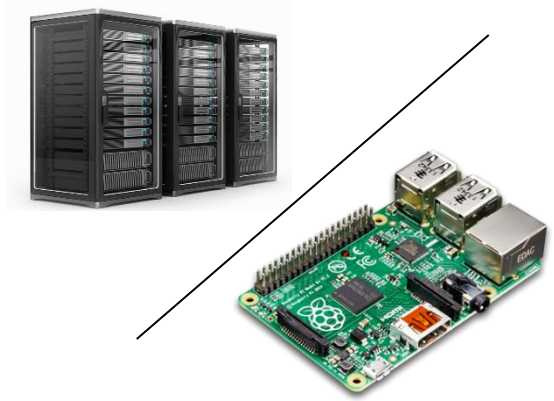
```
if (valid_login == true) {  
    send(user_account_info);  
} else {  
    send(wrong_password_message);  
}
```



Database

(program living on the server)

Built with server code in
Node.js, Ruby, Python, etc.



Server
(Could be your Raspberry Pi)

Program to live on server



Database
(program living on the server)

Two types of database

- Relational Database
- Non-Relational Database

Relational Databases

- Tables data format
- SQL
 - “Structured Query Language”
 - The Language to **query** from table
- MySQL, SQL Server, PostgreSQL, Oracle, etc...
 - Relational Database Management System
 - Use their own “dialect” of SQL
- CS 564

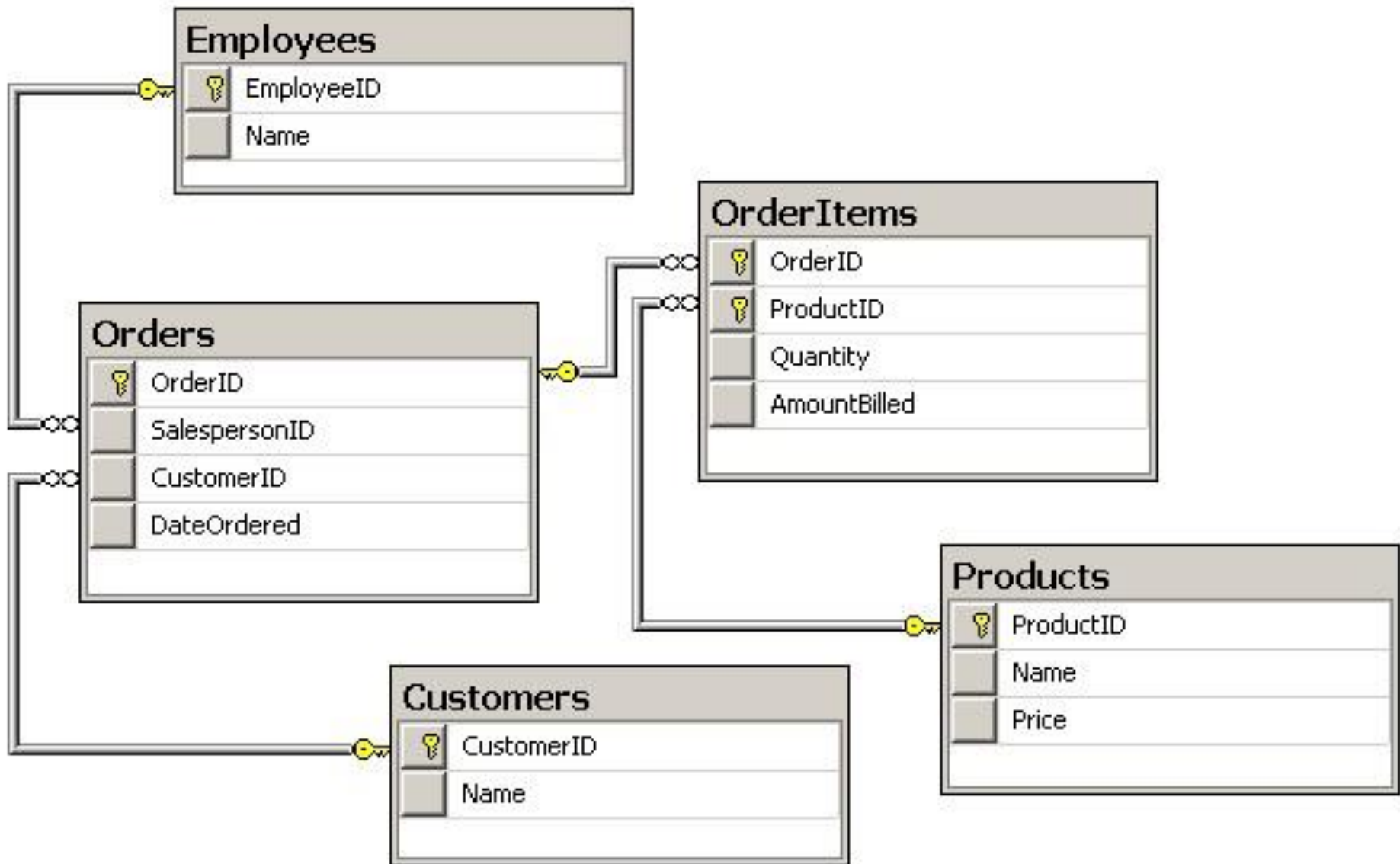
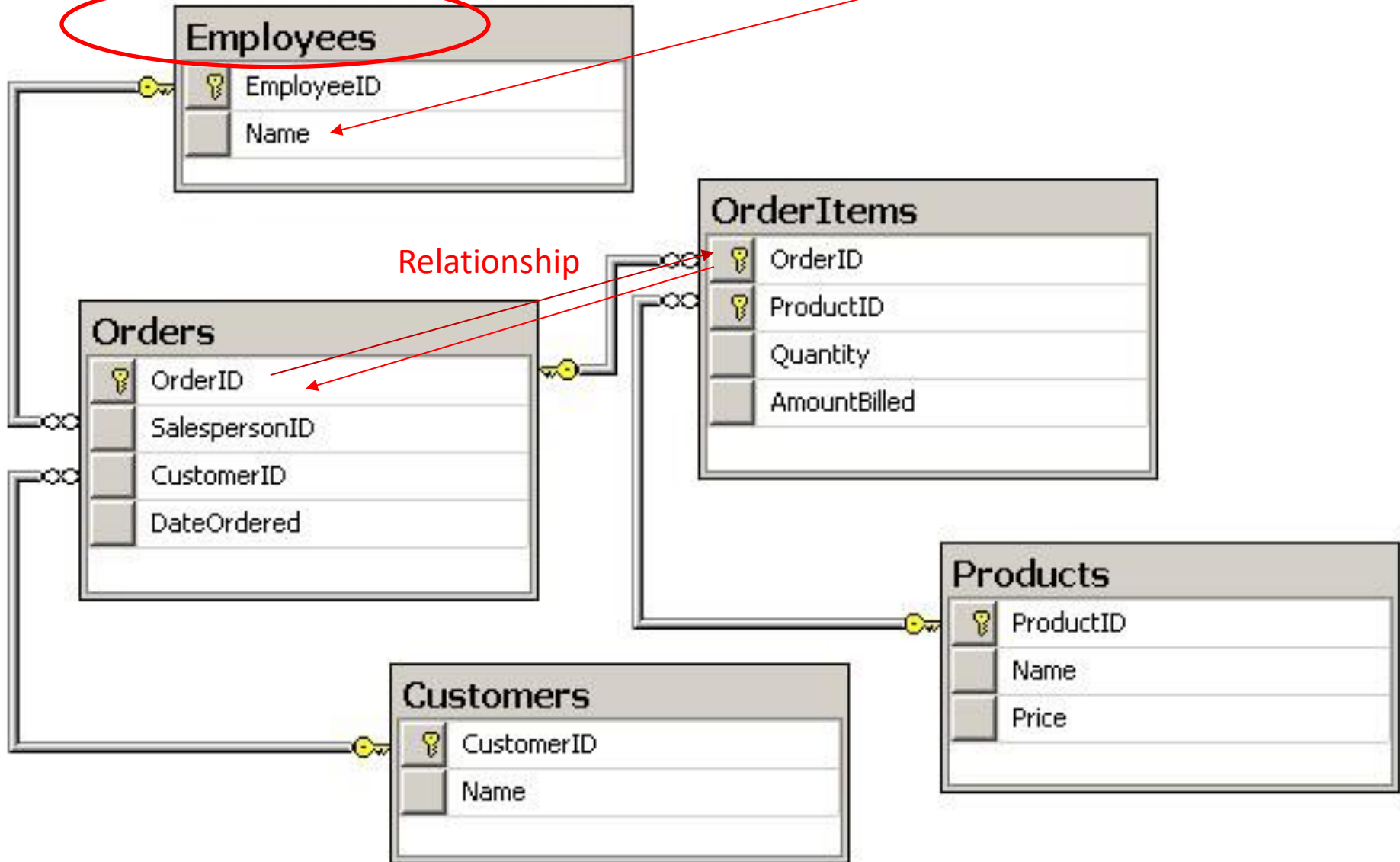


Table Name

Column Name

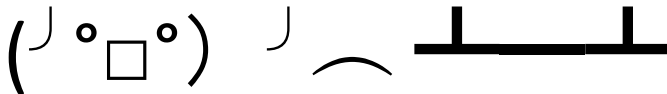
Relationship



SQL Table Example

- http://www.w3schools.com/sql/trysql.asp?filename=try_sql_select_all
- `SELECT * FROM Customers;`
- `SELECT City, ContactName FROM Customers WHERE Country='Mexico';`
- `UPDATE Customers SET ContactName='Alfred Schmidt', City='Hamburg' WHERE CustomerName='Alfreds Futterkiste';`
- `SELECT Customers.CustomerName, Orders.OrderID FROM Customers INNER JOIN Orders ON Customers.CustomerID=Orders.CustomerID ORDER BY Customers.CustomerName;`

Non-Relational

- No More Tables! 
- Also called “NoSQL”
- Can scale “easier” than SQL
 - Don’t need to stick to Schema
 - Might eat these words if not careful
- Can get started right away, no setup
- Web 2.0 choice of database
 - Facebook, Google, Amazon, etc.

What is MongoDB

- A “NoSQL” Database
- Non-Relational
- Free
- Easy to install
 - Just have to type:
`sudo apt-get install mongodb-server`
- “A big JSON garbage bin”
 - Only if you are not careful

JSON Overview

- “Javascript Object Notation”
- Quick Demo!

- Select all and copy
- Type: **var data =**
 - Then paste (ctrl+v) data in console and hit enter
- Undefined?
 - Type: **data**

NOTE: There is never a comma after last listed item

- Can use built-in JavaScript
`<ArrayName>.find(function(e){return e.name == "Alakazam"})`

Time to get to the Pis

- Switch internet to the “**IEEE**” network
 - Password is: **ieeeiscool**
- Use SSH (Mac) or Putty (Windows) to connect
 - `ssh pi@192.168.1.xxx` (← Macs)
 - `pi@192.168.1.xxx` (← Windows)

Mongod vs Mongo ?

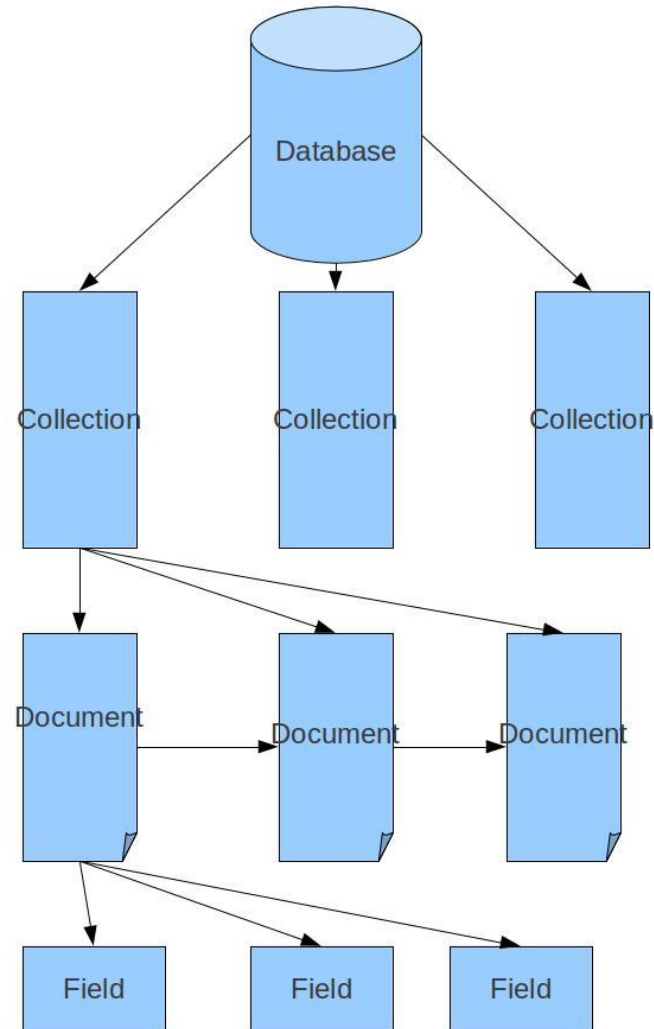
- **Mongod** is “Running MongoDB”
 - Stands for “Mongo Daemon”
- **Mongo** is the command line shell interface which we config with
 - To access it just type “mongo”
- Other options include **mongoimport** and **mongoexport** (will use later)

Linux Services

- Check your Mongo Service by typing:
`sudo Service mongodb status`
- Other operations
`sudo Service mongodb start`
`sudo Service mongodb stop`
`sudo Service mongodb restart`

Mongo

- Database
 - Using only 1
- Collections
 - One big JSON value
- Documents
 - Each part of the JSON
- Field
 - The data you want



- **Database:** CollegeOfEngineering
 - **Collection:** Professors
 - **Collection:** Buildings
 - **Collection:** Students
 - **Documents:**
 - { Name : “Spencer”, Age : 21, Type : “ECE” }
 - { Name : “Fred”, Age : 24, Type : “ME” }
 - { Name : “Willy”, Age : 19, Type : “BME” }

Name	Age	Type
Spencer	21	ECE
Fred	24	ME
Willy	19	BME

» **Field:** “Spencer”

» **Field:** 24

Mongo Shell

- To run type: **mongo**
 - **ctrl+c** to exit the shell
- Start by setting the Database we are going to use
 - Lets call it **Students**
 - Type: **use Students**
 - Sets variable “**db**” to the database
- To see all Databases
 - Type: **show dbs**
- Will not create one until written too

Add some data

Name	Age	Type
Spencer	21	ECE
Fred	24	ME
Willy	19	BME

- Will put in Collection **Engineers**
 - `db.Engineers.insert(
 {
 Name : "<insertYourName>" ,
 Age : "<insertYourAge>",
 Type : "<insertYourType>"
 }
)`

NOTE: No comma after your Type

Other ways to insert

- Set a variable

```
student2 = { Name : "Fred" }
```

```
db.Engineers.insert(student2)
```

– HINT: Tab completes still works in Mongo shell

- What happens if you do the insert twice?

Read your data

- Use different “Find” methods to query data

```
db.Engineers.find()
```

```
db.Engineers.find( {Name : “Fred”} )
```

```
db.Engineers.findOne( {Name : “Fred”} )
```

- Sort it as well

```
db.Engineers.find().sort( {Name: 1 } )
```

- Use -1 for descending order

- Count

```
db.Engineers.count()
```

```
db.Engineers.find( {Name : “Fred”} ).count()
```

Messed up? Just Update!

- Update() takes 3 parameters
 - Query to find the document you want
 - What to update with it
 - Additional options (optional parameter)

- First we will add information

```
db.Engineering.update(  
    { Name : "Fred" },  
    {  
        $set : { Age : 23, Type: "ME" }  
    }  
)
```

- Works the same to edit information

```
db.Engineering.update(  
    { Name : "Fred" },  
    {  
        $set : { Age : 24 }  
    }  
)
```

Need More Data!

- Can take raw JSON objects and import them
 - Can also export collections to JSON files
 - Perform **outside** the mongo shell
- `mongoimport --db World --collection Population --drop --file Population.json`
- `mongoimport --db World --collection NobelPrize --drop --file NobelPrize.json`
- `mongoimport --db World --collection Companies --drop --file Companies.json`
 - Save time and use the `./importScript`

Queries

- Don't forget to use World
- show collections
- `db.Population.find({ males : { $gt : 2250000 } })`
- Too much information?
 - Add 2nd parameter to find to limit fields
 - Uses 1 and 0 to turn on or off
 - `_id` will be set to 1 by default
- `db.Population.find(`
`{ males : { $gt : 2250000 } },`
`{ males : 1, age : 1, _id : 0 }`
`)`

Find the order of the most females in the 30-39 age range

- Easy!
- ```
db.Population.find({ age : {
 $gte : 30, $lte : 39
 }
 },
 {age : 1, females : 1, _id:0}
).sort(
 { females : -1 }
)
```

When has there been only two Nobel Prize Laureates and one of them was named either John, Bob, and/or Eric?

- Oddly specific, talk about being bored
- `db.NobelPrize.find(  
 {  
 "laureates.firstname" : {  
 $in: ["John", "Bob", "Eric"]  
 },  
 laureates : {$size: 2}  
 },  
 {year : 1, category:1, _id:0}  
)`

# Moral of the story

- Use a GUI if possible
- My choice is MongoChef
  - Note: Free for non commercial use
  - Has built in import and export functions as well



Time to get some practice!

**[tinyurl.com/ServerFunIEEE](https://tinyurl.com/ServerFunIEEE)**