# WEBD 236

## Web Information Systems Programming

## Week 9


Todd Whittaker and Scott Sharkey
(sharkesc@franklin.edu)

FRANKLIN UNIVERSITY

# Agenda

- This week’s expected outcomes
- This week’s topics
- This week’s homework
- Upcoming deadlines
- Questions and answers

# Week 9 Outcomes

- Design databases from real-world problem statements
- Normalize databases
- Employ SQL to create database tables and indices
- Employ PHP scripting to create and load tables with initial data
- Send e-mail from web applications.
- Examine JSON for data-interchange.

# Database Design

- Six steps to database design
  - Identify the data elements
  - Subdivide each element into its smallest useful components
  - Identify the tables and assign columns
  - Identify primary and foreign keys (relationships)
  - Normalize
  - Identify indices

# Database Design

- "Things" to remember
  - Each table represents a group of "things"
  - Each row within the table is one "thing" in the group
  - Each column within a row represents an attribute of that one "thing"

# Database Design

- Example:
  - Consulting company connects people with the right skills to clients who need those people to work on specific projects
  - Need to know which employees have certain skills (and at what level), and assign them to a project for certain dates.
  - A client can have more than one project, but each project has a specific contact person (sponsor) within the client's organization.

# Database Design

- Identify the major entities

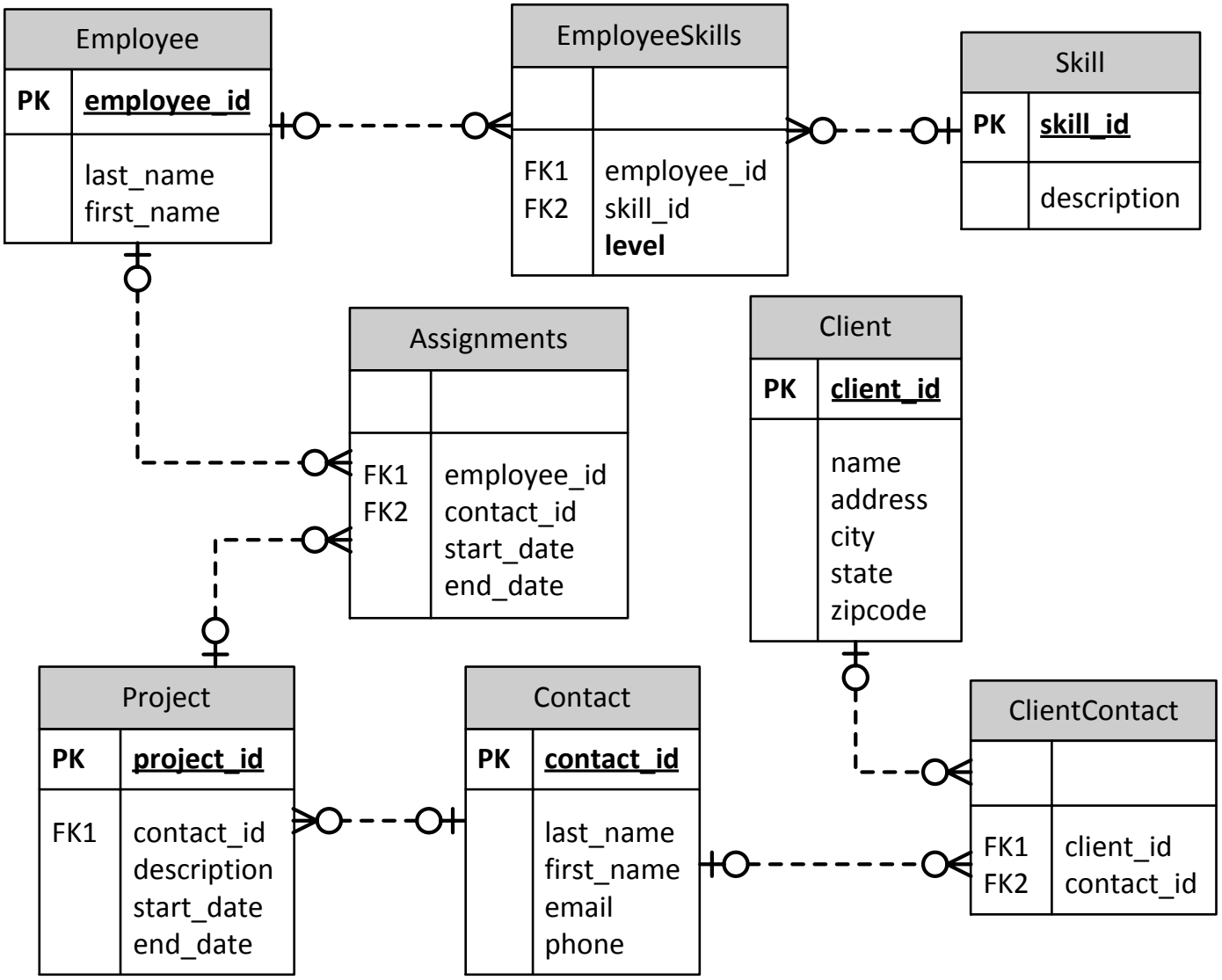
# Database Design

- Identify the attributes of each entity

# Database Design

- Draw the ERD

# Database Design



# Database Design

- Is the database normalized?
  - **First normal form**: tabular format, no repeating groups, primary key identified, non-key attributes are dependent on primary key.
  - **Second normal form**: In 1NF and no partial dependencies (no dependencies on just part of the key).

# Database Design

- Is the database normalized?
  - **Third normal form**: 2NF and every non-key attribute must depend only on the primary key
  - **Boyce-Codd normal form:** (BCNF) a non-key attribute cannot depend on a non-key attribute (avoids transitive dependencies)
  - **Higher forms:** interesting, but not particularly useful for this course.

# Database Design

- Is the database normalized?
  - If the DB is not in at least 3rd normal form, what good reason do you have for doing so?

# Database Design

- What should we create indices for?
  - Primary keys
  - Foreign keys
  - Use frequently in search or join (see above)

# Database Design

- Tools for modeling/designing databases
  - DB vendors usually provide excellent tools
    - MySQL: MySQL Workbench
    - Oracle: SQL Developer Data Modeler
    - Microsoft: Visio

You have access to this for free through MSDNAA:
http://msdn07.e-academy.com/FU_DC

# SQL for Creating Databases

- CREATE DATABASE – Needed in a multi-database server (MySQL), but not for SQLite.

```
CREATE DATABASE IF NOT EXISTS Contracting;
```

- USE – again, for servers w/many DBs

```
USE Contracting;
```

- DROP DATABASE – irreversible

```
DROP DATABASE Contracting;
```

# SQL for Creating Databases

- CREATE TABLE for Client

| Client | |
|---|---|
| **PK** | **client_id** |
| | name<br>address<br>city<br>state<br>zipcode |

```
CREATE TABLE IF NOT EXISTS Client {
  client_id INTEGER      PRIMARY KEY NOT NULL, /* auto-inc */
  name      VARCHAR(100) NOT NULL,
  address   VARCHAR(60)  NOT NULL,
  city      VARCHAR(30)  NOT NULL,
  state     CHAR(2)      NOT NULL,
  zipcode   VARCHAR(10)  NOT NULL
}
```

Other common, types: DECIMAL, TEXT, CLOB, BLOB, DATE, TIME, DATETIME

# SQL for Creating Databases

- CREATE TABLE for EmployeeSk

| EmployeeSkills | |
|---|---|
| | |
| FK1<br>FK2 | employee_id<br>skill_id<br>**level** |

```
CREATE TABLE IF NOT EXISTS EmployeeSkills {
  employee_id INTEGER NOT NULL,
  skill_id    INTEGER NOT NULL,
  level       INTEGER NOT NULL,
  FOREIGN KEY(employee_id) REFERENCES Employee(employee_id),
  FOREIGN KEY(skill_id)    REFERENCES Skill(skill_id)
}
```

Remember, Employee and Skill need to exist before this command is executed.

# SQL for Creating Databases

- DROP TABLE

```
DROP TABLE IF EXISTS Client;
```

- CREATE INDEX

```
CREATE UNIQUE INDEX email ON Contact(email);
```

- DROP INDEX

```
DROP INDEX email ON Contact;
```

# SQL for Creating Databases

- Privileges
  - Individual users in the DB server can have rights to perform certain actions:
    - Globally
    - On certain databases
    - On certain tables within databases
    - On certain columns within tables
  - CREATE, ALTER, DROP, INDEX, SELECT, INSERT, UPDATE, DELETE

# SQL for Creating Databases

- Privileges
  - Reality: at best two users at the DB level
    - One that can do everything
    - One that is read-only (for a data warehouse)
  - In a web application, privileges are usually enforced at the *application* level in *code*.
    - Approaches:
      - RBAC: role based access control
      - ACL: access control lists
      - DAC: discretionary access control

More on this in Ch. 21

# Loading Initial Data

- Why load initial data
  - Need the DB to be in an initial state for the application to work
  - Need something to test interactively

```
Joe,O'Conner,joeo@example.com
John,Roberts,johnr@example.com
Bill,Kilbourne,billk@example.com
Ruth,Richardson,ruthr@example.com
```

data.txt file containing contact information to load

# Loading Initial Data

```
class Db {
    protected static function getDb() {
        try {
            $db = new PDO('sqlite:contacts.db3');
        } catch (PDOException $e) {
            die("Could not open database. " .
                $e -> getMessage());
        }
        return $db;
    }

    // continued...
```

# Loading Initial Data

```
class Db {
    public static function resetTable($class) {
        $reflector = new ReflectionClass($class);

        $db = self::getDb();
        $db -> beginTransaction();

        $db -> exec('DROP TABLE IF EXISTS '.
            $reflector -> getConstant('TABLE_NAME'));
        $db -> exec($reflector ->
            getConstant('CREATE_TABLE_SQL'));
        $db -> commit();
    }

    // continued...
```

# Loading Initial Data

```
class Db {
    public static function resetTable($class) {
        $reflector = new ReflectionClass($class);

        $db = self::getDb();
        $db -> beginTransaction();

        $db -> exec('DROP TABLE IF EXIS
            $reflector -> getConstant('TABL
        $db -> exec($reflector ->
            getConstant('CREATE_TABLE_
        $db -> commit();
    }

    // continued...
```

ReflectionClass lets you find out information about a class at runtime (i.e. a list of methods, properties, constants, etc.)

# Loading Initial Data

```
class Db {
    public static function loadInitial($fileName, $class) {

        $fp = @fopen($fileName, "r");
        if ($fp) {
            while (($line = fgets($fp)) !== false) {
                $args = explode(",", $line);
                $reflector = new ReflectionClass($class);
                $object = $reflector -> newInstanceArgs($args);
                $object -> insert();
            }
            fclose($fp);
        }
    }

    // continued...
```

# Loading Initial Data

```
class Db {
    public static function loadInitial($fileName, $cla

        $fp = @fopen($fileName, "r");
        if ($fp) {
            while (($line = fgets($fp)) !== false) {
                $args = explode(",", $line);
                $reflector = new ReflectionClass($cl
                $object = $reflector -> newInstanceArgs(
                $object -> insert();
            }
            fclose($fp);
        }
    }

    // continued...
```

fopen opens a file (duh).

fgets reads a string from the file up to a newline or eof.

# Loading Initial Data

```
class Model extends Db {
    protected $id;

    public function __construct($id = null) {
        if ($id) {
            $this -> setId($id);
        }
    }
    public function getId() {
        return $this -> id;
    }
    public function setId($id) {
        $this -> id = $id;
        return $this;
    }
}
```

# Loading Initial Data

```
class Contact extends Model {
    protected $first_name;
    protected $last_name;
    protected $email;

    const TABLE_NAME = 'Contacts';

    const CREATE_TABLE_SQL = '
      CREATE TABLE Contacts (
        contact_id INTEGER PRIMARY KEY,
        first_name VARCHAR(20) NOT NULL,
        last_name  VARCHAR(30) NOT NULL,
        email VARCHAR(50) NOT NULL
      );';

 // continued...
```

# Loading Initial Data

```
class Contact extends Model {

  const INSERT_SQL = '
    INSERT INTO Contacts
      (first_name, last_name, email)
    VALUES
      (:first_name, :last_name, :email)';

   public function __construct($first, $last, $email) {
      $this -> first_name = $first;
      $this -> last_name = $last;
      $this -> email = $email;
   }

 // continued...
```

# Loading Initial Data

```
class Contact extends Model {

    public function insert() {
        $db = self::getDb();
        $st = $db -> prepare(self::INSERT_SQL);
        $st -> bindParam(":first_name", $this -> first_name);
        $st -> bindParam(":last_name", $this -> last_name);
        $st -> bindParam(":email", $this -> email);
        $st -> execute();
        $this -> id = $db -> lastInsertId();
    }
}

// and then somewhere...
Db::resetTable("Contact");
Db::loadInitial("data.txt", "Contact");
```

# Sending email from PHP

- SMTP (Simple Mail Transfer Protocol)
  - Must have a properly configured SMTP server.
  - Easy to configure it to work with Google's SMTP servers:
    - Create a GMail account, enable POP.
    - Communicate with Google over SSL.

# Sending email from PHP

- Configuration issues with XAMPP
  - Must turn on SSL by adding the following line to C:\xampp\php\php.ini

```
; enable SSL for e-mail send/receive
extension=php_openssl.dll
```

  - Then restart Apache

# Sending email from PHP

- Using the PEAR (PHP Extension and Application Repository) library for mail
  - Require the 'Mail.php' file, pulled from c:\xampp\php\PEAR
  - Turn off strict error reporting (Mail.php is a library written for PHP4)

```
<?php
error_reporting(E_ALL & !E_STRICT);
require_once 'Mail.php';
```

# Sending email from PHP

- Let's abstract this into a class!

```
class Email {
    private static $smtpHost = "ssl://smtp.gmail.com";
    private static $smtpPort = 465;
    private static $smtpAuth = true;
    private static $smtpUsername = 'someaccount@gmail.com';
    private static $smtpPassword = 'someAccountPassword';
    private $smtp;
    private $recipients;
    private $carbonCopy;
    private $blindCopy;
    private $subject;
    private $message;
    private $sender;
    private $body;
    private $contentType;
```

# Sending email from PHP

- Let's abstract this into a class!

```
public function __construct() {
    $options = array();
    $options['host'] = self::$smtpHost;
    $options['port'] = self::$smtpPort;
    $options['auth'] = self::$smtpAuth;
    $options['username'] = self::$smtpUsername;
    $options['password'] = self::$smtpPassword;

    $mail = new Mail();
    $this -> smtp = $mail -> factory('smtp', $options);

    if (is_a($this -> smtp, 'PEAR_Error')) {
        throw new Exception("Could not create mailer");
    }
    // ...continued...
```

# Sending email from PHP

- Let's abstract th

```
public function __construct
    $options = array();
    $options['host'] = self::$sm
    $options['port'] = self::$sm
    $options['auth'] = self::$sm
    $options['username'] = se
    $options['password'] = self::$smtp

    $mail = new Mail();
    $this -> smtp = $mail -> factory('smtp', $options);

    if (is_a($this -> smtp, 'PEAR_Error')) {
        throw new Exception("Could not create mailer");
    }
    // ...continued...
```

Your textbook uses Mail::factory(), but that triggers errors in E_STRICT mode since factory isn't declared to be static.

Same here where your textbook uses PEAR::isError().

# Sending email from PHP

- Let's abstract this into a class!

```
        $this -> recipients = array();
        $this -> carbonCopy = array();
        $this -> blindCopy = array();
        $this -> sender = '"WEBD236 Team" <' .
            self::$smtpUsername . '>';
    }

    public function addRecipient($recipient) {
        $this -> recipients[] = $recipient;
    }

    public function setRecipient($recipient) {
        $this -> recipients = array();
        $this -> addRecipient($recipient);
    }
```

# Sending email from PHP

- Let's ab

```
    $this -> recip
    $this -> carb
    $this -> blind
    $this -> send
        self::$smt
}

public function a
    $this -> recipients[] = $
}

public function setRecipient($recipient) {
    $this -> recipients = array();
    $this -> addRecipient($recipient);
}
```

The use here is that you can set many different options, and then from within a loop call setRecipient() followed by send() to deliver customized email messages. BCC is more efficient if the message is the same (one send vs. many).

# Sending email from PHP

- Let's abstract this into a class!

```
public function addCC($recipient) {
    $this -> carbonCopy[] = $recipient;
}
public function setCC($recipient) {
    $this -> carbonCopy = array();
    $this -> addCC($recipient);
}
public function addBcc($recipient) {
    $this -> blindCopy[] = $recipient;
}
public function setBcc($recipient) {
    $this -> blindCopy = array();
    $this -> addBcc($recipient);
}
```

# Sending email from PHP

- Let's abstract this into a class!

```
public function setSender($sender) {
    $this -> sender = $sender;
}

public function setSubject($subject) {
    $this -> subject = $subject;
}

public function setBody($body) {
    $this -> body = $body;
}

public function setContentType($contentType) {
    $this -> contentType = $contentType;
}
```

# Sending email from PHP

- Let's abstract this into a class!

```
public function setSender($sender) {
    $this -> sender = $sender;
}

public function setSubject($subject
    $this -> subject = $subject;
}

public function setBody($body) {
    $this -> body = $body;
}

public function setContentType($contentType) {
    $this -> contentType = $contentType;
}
```

Defaults to plain text with no content type. Use "text/html" to send HTML-based mail.

# Sending email from PHP

- Let's abstract this into a class!
  - Wouldn't it be nice to use the same template rendering engine we used for web-pages to generate email templates too?
    - A small, backwards compatible change to renderTemplate in include/util.inc makes this possible.

# Sending email from PHP

- Modifying Lib/Util.inc

```
function renderTemplate($view, $params = array(),
        $asString = false) {
    $useCaching = false; // turn off caching
    if (!file_exists($view)) {
        die("File $view doesn't exist.");
    }
    // do we have a cached version?
    clearstatcache();
    $cacheName = __cacheName($view);
    if ($useCaching && file_exists($cacheName) &&
            (filemtime($cacheName) >= filemtime($view))) {
        $contents = file_get_contents($cacheName);
    } else {

    // ...continued...
```

# Sending email from PHP

- Modifying Lib/Util.inc

```
    $contents = __importTemplate(array('unused', $view));
    $contents = preg_replace_callback('/@@\s*(.*)\s*@@/U',
        '__resolveRelativeUrls', $contents);
    $patterns = array(
        array('src' => '/{{/', 'dst' => '<?php echo('),
        array('src' => '/}}/', 'dst' => ');?>'),
        array('src' => '/\[\[/', 'dst' => '<?php '),
        array('src' => '/\]\]/', 'dst' => '?>')
    );
    foreach ($patterns as $pattern) {
        $contents = preg_replace($pattern['src'],
            $pattern['dst'], $contents);
    }
    file_put_contents($cacheName, $contents);
} // ...continued...
```

# Sending email from PHP

- Modifying Lib/Util.inc

```
    extract($params);
    ob_start();
    eval("?>" . $contents);
    $result = ob_get_contents();
    ob_end_clean();
    if (!$asString) {
        echo $result;
    }
    return $result;
}
```

# Sending email from PHP

- Let's abstract this into a class!
  - Now, if we pass a third parameter (true), then it won't produce output, but will return the rendered string.

# Sending email from PHP

- Meanwhile, back in Email.inc…

```
public function send($template=false, $variables=false) {
  if ($template) {
     $this -> body = renderTemplate($template,
        $variables, true);
  }
  $headers = array();
  $headers['From'] = $this -> sender;
  $headers['To'] = implode(", ", $this -> recipients);
  $headers['Cc'] = implode(", ", $this -> carbonCopy);
  $headers['Bcc'] = implode(", ", $this -> blindCopy);
  $headers['Subject'] = $this -> subject;
  if ($this -> contentType) {
     $headers['Content-type'] = $this -> contentType;
  }
  // ...continued...
```

# Sending email from PHP

- Meanwhile, back in Email.inc…

```
        $allRecipients = array_merge($this -> recipients,
            $this -> blindCopy, $this -> carbonCopy);

        $result = $this -> smtp -> send(implode(", ",
            $allRecipients), $headers, $this -> body);

        if (is_a($result, 'PEAR_Error')) {
            throw new Exception($result -> getMessage());
        }
    }
} // end Email class
```

# Sending email from PHP

- A simple email_template.html

```
<!DOCTYPE html>
<html>
  <head><title>{{$title}}</title></head>
  <body>
    <p>Hello {{$firstName}},</p>
    <p>I'm writing to you today to tell you about our
    great product! It's not Spam, I assure you. It's an
    e-mail system that integrates easily with PHP and it's
    built on top of PEAR::Mail. Please consider buying it.</p>
    <p>Sincerely,</p>
    <p>WEBD236-V1WW</p>
  </body>
</html>
```

# Sending email from PHP

- Using the Email class

```
$subject = "Testing an e-mail through PHP";
$recipient = '“Scott Sharkey" <sharkesc@franklin.edu>';
$email = new Email();
$email -> setRecipient($recipient);
$email -> setSubject($subject);
$email -> setContentType('text/html');
$email -> send(
    'email_template.html',
    array(
        'title' => $subject,
        'firstName' => ‘Scott'
    )
);
```

# Sending email from PHP

- Using

```
$subject = "Testi
$recipient = "'To
$email = new Em
$email -> setRec
$email -> setSub
$email -> setCon
$email -> send(
    'email_templa
    array(
        'title' => $subject,
        'firstName' => 'Todd'
    )
);
```

**Testing an e-mail through PHP** March 28, 2012 10:53 AM

From: "WEBD 236" <webd236@gmail.com>

To: "Todd Whittaker" <todd.whittaker@franklin.edu>

Hello Todd,

I'm writing to you today to tell you about our great product! It's not Spam, I assure you. It's an e-mail system that integrates easily with PHP and it's built on top of PEAR::Mail. Please consider buying it.

Sincerely,

WEBD236-V1WW

# Sending email from PHP

- Covered more in the textbook
  - Email address validation with RFC822

# Accessing remote data

- It is possible to pull data from any web site
  - But, it requires parsing the HTML
  - XML is easier to parse, but still cumbersome
  - JSON (JavaScript Object Notation) is designed for easy parsing.
    - Rapidly becoming the back-end of all web sites
    - “One page apps” like Google Mail send and receive JSON to communicate with the back end, and use JavaScript on the front end to display results.

# Accessing remote data

- It is possible to pull data from any web site
  - But, it
  - XML i umbersome
  - JSON n) is designed f asy parsing.
    - Rapidly becoming the back-end of all web sites
    - “One page apps” like Google Mail send and receive JSON to communicate with the back end, and use JavaScript on the front end to display results.

AJAX (Asynchronous JavaScript and XML) has really become AJAJ (Asynchronous JavaScript and JSON).

# Accessing remote data


```
[{
    "trends" : [{
        "url" : "http:\/\/twitter.com\/search\/%23IfIMeetJustinBieber",
        "query" : "%23IfIMeetJustinBieber",
        "name" : "#IfIMeetJustinBieber",
        "events" : null,
        "promoted_content" : null
    }, {
        "url" : "http:\/\/twitter.com\/search\/%23YouKnowItsFriday",
        "query" : "%23YouKnowItsFriday",
        "name" : "#YouKnowItsFriday",
        "events" : null,
        "promoted_content" : null
    }, {
        "url" : "http:\/\/twitter.com\/search\/%23WeLoveKevinJonas",
        "query" : "%23WeLoveKevinJonas",
        "name" : "#WeLoveKevinJonas",
        "events" : null,
        "promoted_content" : null
    }, {
    // ...snip...
    }],
    "created_at" : "2012-03-28T17:37:59Z",
    "as_of" : "2012-03-28T17:38:26Z",
    "locations" : [{
        "name" : "Worldwide",
        "woeid" : 1
    }]
}]
```


Raw JSON returned from Twitter

```
array(1) {
  [0]=>
  object(stdClass)#2 (4) {
    ["trends"]=>
    array(10) {
      [0]=>
      object(stdClass)#3 (5) {
        ["url"]=>
        string(48) "http://twitter.com/search/%23IfIMeetJustinBieber"
        ["query"]=>
        string(22) "%23IfIMeetJustinBieber"
        ["name"]=>
        string(20) "#IfIMeetJustinBieber"
        ["events"]=>
        NULL
        ["promoted_content"]=>
        NULL
      }
      [1]=>
      object(stdClass)#4 (5) {
        ["url"]=>
        string(45) "http://twitter.com/search/%23YouKnowItsFriday"
        ["query"]=>
        string(19) "%23YouKnowItsFriday"
        ["name"]=>
        string(17) "#YouKnowItsFriday"
        ["events"]=>
        NULL
        ["promoted_content"]=>
        NULL
      }
    // ...snip...
    }
}
```

JSON converted to PHP arrays/objects via json_decode().

# Accessing remote data

- Considerations when building a web app
  - JSON is the data-interchange format of the web.
  - If you want to build something that others build on, then you'll need to expose services using JSON like Twitter does.
  - Page-refresh apps (like we've been building) are old technology. Ajax and single-page apps take less bandwidth, use REST more effectively, and are more responsive to users.

# Upcoming Deadlines

- Readings for next week
  - Chapters 18 and 19 in *PHP and MySQL*
- Assignments
  - Homework 8 due end of week 11
  - Lab 4 due end of week 12
- Next week:
  - Role-based Access Control Lists

# General Q & A

- Questions?
- Comments?
- Concerns?

FRANKLIN UNIVERSITY