Lecture #28: This is the end - the only end my friends.
Administrivia

- Homework
  - HW 9 Is Out
- Do Some Reading
  - Check Out Python and MongoDB Book
On The Menu

- Review
  - MySQL Syntax
  - PHP
  - PHP + MySQL
  - SQLite
  - MongoDB
Create Table Syntax

CREATE TABLE [table_name] (  
  [column_name] [column_type] [column_details],  
  KEY([column_name(s)]),  
  UNIQUE KEY([column_name(s)]),  
  PRIMARY KEY([column_name(s)]),  
  FOREIGN KEY ([column_name]) REFERENCES  
  [table_name]([column_name])  
)  
ENGINE=[engine]  
DEFAULT CHARSET=[charset]  
COLLATE=[collation];
INSERT Syntax

INSERT INTO
    [table_name] (column_1, column_2,...)
VALUES
    (value_1, value_2,...);
**INSERT Syntax**

```
INSERT INTO
    [table_name] (column_1, column_2,...)
VALUES
    (value_1, value_2,...),
    (value_1, value_2,...),
    (value_1, value_2,...),
    (value_1, value_2,...),
    (value_1, value_2,...);
```
UPDATE Syntax

UPDATE [table_name]
SET
  [column_1] = [value_1]
  [column_2] = [value_2]
WHERE
  [condition_1]
  [(AND|OR) condition_2]
  [(AND|OR) condition_3]
  ...
  [(AND|OR) condition_n]
DELETE Syntax

DELETE FROM [table_name]
WHERE
  [condition_1]
  [(AND|OR) condition_2]
  [(AND|OR) condition_3]
  ...
  [(AND|OR) condition_n]
Simple Select Statement

SELECT  
   *  
FROM    
   table  
LIMIT x  
OFFSET y
Simple Select Statement

SELECT
    row_1, row_2,...
FROM
    table_1, table_2,...
WHERE
    condition_1 (AND | OR)
    condition_2 (AND | OR)
    ...

Simple Select Statement with order by

```
SELECT
    row_1, row_2,...
FROM
    table_1, table_2,...
WHERE
    condition_1 (AND | OR)
    condition_2 (AND | OR)
ORDER BY
    column_1,
    column_2
```
Aggregate Queries

SELECT
    SUM(numeric_column_1),
    MIN(numeric_column_2),
    MAX(numeric_column_3),
    AVG(numeric_column_4),
    COUNT(any_column_5),
    COUNT(DISTINCT any_column_1),
FROM
    table_1
...
GROUP BY

- A SELECT against a table generates a “virtual” table containing the matching rows.
- Adding a GROUP BY [col_name] statement collapses this virtual table so that there is just one row for each individual value of [col_name].
- GROUP BY statements are often combined with aggregate functions (SUM, MIN, MAX, AVG).
Aggregate Queries With Group By

SELECT
    SUM(numeric_column_1),
    other_column_1,
FROM
    table_1
GROUP BY
    other_column_1
JOIN

- A SELECT against a table generates a “virtual” table containing the matching rows.
  - E.G. `SELECT * FROM t1;`
- A SELECT against multiple tables generates a “virtual” table containing combined rows from both tables.
  - E.G. `SELECT * FROM t1 JOIN t2;`
- JOINS allow us to specify how we want rows from different to be combined in a SELECT over multiple tables.
JOIN

- `SELECT * FROM professor;`
- `SELECT * FROM class;`
- `SELECT * FROM professor JOIN class;`
  - Returns ALL POSSIBLE combinations of rows in the professor and class tables.
  - If we only want to see the combination of rows that reflect which professor is teaching which class, we must specify some type of join.
INNER JOIN

- When SELECTING an inner join between tables \( t_1 \) and \( t_2 \) on key \( k \) the only rows that are returned are those rows in which \( t_1.k = t_2.k \).

  - SELECT * FROM professor JOIN CLASS ON professor.professor_id = class.professor_id;
INNER JOIN - Visualized

INNER JOIN

Table1

SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
ON t1.Col1 = t2.Col1

(C) http://blog.SQLAuthority.com
INNER JOIN

- Lots of different ways of expressing - the following are equivalent in MySQL
  - \texttt{SELECT * FROM t1 INNER JOIN t2 ON t1.k = t2.k;}
  - \texttt{SELECT * FROM t1 JOIN t2 ON t1.k = t2.k;}
  - \texttt{SELECT * FROM t1, t2 WHERE t1.k = t2.k;}

NYU
OUTER JOIN

- Outer joins come in three flavors.
  - LEFT
  - RIGHT
  - FULL
LEFT OUTER JOIN

- When SELECTING a LEFT OUTER JOIN from tables t1 and t2 on key k the result contains:
  - All rows from t1
  - All rows from t2 that match t1 on k or NULL if no match exists.
- SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.k=t2.k;
LEFT OUTER JOIN

SELECT *
FROM Table1 t1
LEFT OUTER JOIN Table2 t2
ON t1.Col1 = t2.Col1

(C) http://blog.SQLAuthority.com
RIGHT OUTER JOIN

- When SELECTING a RIGHT OUTER JOIN from tables t1 and t2 on key k the result contains:
  - All rows from t2
  - All rows from t1 that match t2 on k or NULL if no match exists.

- SELECT * FROM t1 RIGHT OUTER JOIN t2 ON t1.k=t2.k;
RIGHT OUTER JOIN

SELECT *
FROM Table1 t1
RIGHT OUTER JOIN Table2 t2
ON t1.Col1 = t2.Col1

(C) http://blog.SQLAuthority.com
FULL OUTER JOIN

- When SELECTING a FULL OUTER JOIN from tables t1 and t2 on key k the result contains:
  - All rows from t2
  - All rows from t1
  - NULL values for t1 and t2 rows where there exists no match on key k.

- No straightforward way of typing this in MySQL
  - USE Union instead
FULL OUTER JOIN

SELECT *
FROM Table1 t1
FULL OUTER JOIN Table2 t2
ON t1.Col1 = t2.Col1

(C) http://blog.SQLAuthority.com
PHP

- PHP = The **PHP Hypertext Preprocessor**
- Invented in 1995
- Server Side Scripting Language (deets on next slide)
- Running on 244 Million Websites
- Evan Sandhaus is not a huge fan
PHP - Server Side Scripting Language
PHP - Hello World

```php
<?php
    echo "Hello World!";
?>

OR

<?php
    print "Hello World!";
?>
PHP - Variables

// numbers
$current_year = 2013;
$pi = 3.14159;
$distance_to_moon = 3.844e6;
PHP - Variables

```php
// booleans
$just_true  = true;
$just_false = false;
$or_true    = $just_true || $just_false;  // true
$and_true   = $just_true && $just_true;   // true
```
// Strings
$single_quotes = 'Single Lady';
$double_quotes = "By Beyonce";

// String Concatenation
$concat = $single_quotes." \\
$.double_quotes;
// "Single Lady By Beyonce"

// String Formatting
$formatted_string = "Welcome to $current_year!";
// "Welcome to 2013"

$format_fail = 'Welcome to $current_year!';
// "Welcome to $current_year"

$formatter_comma = 'Welcome to '.number_format($current_year);
// "Welcome to 2,013"
PHP - Arrays

// Arrays

// creates an empty array
$simple_array = array();

// set position 0 to 1
$simple_array[0] = 1;

// create an array with the specified values
$simple_array = array(1, 2, 3, 5, 7, 11, 13);

// append 17 to the end of the array
$simple_array[] = 17;

// assign value 5 to $a_prime
$a_prime = $simple_array[3];

// assign length of $simple_array (8) to $a_length
$a_length = sizeof($simple_array);
// Dictionaries - also arrays
// but with keys that aren't sequential integers

// create an empty array
$simple_dictionary = array();

// assign value 'Engel' to key 'Deena'
$simple_dictionary['Deena'] = 'Engel';

// Inline dictionary declaration
$simple_dictionary = array(
    'Evan' => 'Sandhaus',
    'Tara' => 'Bobiak'
);

// Assigns 'Sandhaus' to $lname
$lname = $simple_dictionary['Evan'];

// Assign "Mr. Sandhaus plans to marry Ms. Bobiak"
// to $array_concat
$array_concat = "Mr. {$simple_dictionary['Evan']} plans to marry Ms. {$simple_dictionary['Tara']}";
$simple_dictionary = array(
    'Evan' => 'Sandhaus',
    'Tara' => 'Bobiak'
);

// Simple foreach loop
// Prints out
// Evan Sandhaus
// Tara Bobiak
foreach ($simple_dictionary as $key => $value) {
    echo "$key $value<br>

}
PHP Built-In Functions

- **is_array(x), is_boolean(x), is_string(x)**: True if x is of the specified type.
- **implode(glue, array)**: Create a string containing each of the elements from array separated by the 'glue' string.
- **isset(array[key])**: Returns **true** if key exists in array, **false** otherwise.
- **sizeof(array), count(array)**: Return the size of an array.
- **http://php.net/quickref.php**
What Is Truth

- When converting to boolean, the following values are considered FALSE:
  - the boolean FALSE itself
  - the integer 0 (zero)
  - the float 0.0 (zero)
  - the empty string, and the string "0"
  - an array with zero elements
  - an object with zero member variables (PHP 4 only)
  - the special type NULL (including unset variables)
PHP Selective Blocks

<html>
<head>
    <title>Insert Demo</title>
</head>
<body>
    <?php if ($condition) : ?>
        <!-- Printed iff $condition is true -->
    <?php elseif($condition_2): ?>
        <!-- Printed iff $condition is false and $condition_2 is true -->
    <?php else: ?>
        <!-- Printed if both $condition and $condition_2 are false. -->
    <?php endif; ?>
</body>
</html>
Form

```html
<form action="cms.php" method="post">
</form>
```
Inputs

<input type="text" name='headline' />

<select name='issue'>
  <option value='winter'>Winter</option>
  <option value='spring'>Spring</option>
  <option value='summer'>Summer</option>
  <option value='fall'>Fall</option>
</select>
Inputs Continued

<input type='radio' name='content_type' value='news'>News</input>
<input type='radio' name='content_type' value='opinion'>Opinion</input>

<textarea rows="10" cols="60" name='article_body'></textarea>

<input type="checkbox" name="tag[]" value="us">United States<br>
<input type="checkbox" name="tag[]" value="world">World<br>
<input type="checkbox" name="tag[]" value="sports">Sports<br>
<input type="checkbox" name="tag[]" value="entertainment">Entertainment<br>
Hidden Inputs In Forms

- `<input type='hidden' value='hidden_value' name='some_name'>`
Submit

<input type="submit" value="Submit"/>
PHP & MySQL

- connect
  - $database_connection = mysqli_connect($host, $username, $password, $database_name);
PHP Mysql

- Query
  - $results = mysqli_query($database_connection, $sql);
PHP Mysql

- **Handle Results**
  - ```
    while ($row = $result->fetch_assoc()) {
      ...
    }
  ```
  - Iterates over all rows returned by the database as an associative array between column names and values.
MYSQLI - More Goodies

- $results->num_rows;
  - Return the number of rows in the result

- $results->fetch_row();
  - Return the results as a numeric array (not associative).

- $results->fetch_array (MYSQLI_ASSOC | MYSQLI_NUM | MYSQLI_BOTH);
  - Return the results as a the specified type of array
A bit more about mysqli_query()

- $results = mysqli_query($database_connection, $sql);
- What is results set to?
  - A mysqli_result for SELECT, SHOW, DESCRIBE or EXPLAIN queries.
  - TRUE for all other successful queries
  - FALSE if the query fails.
PHP Query Security

- PHP will often take user inputs and place those inputs into a query.
- Malicious actors (bad hackers) can embed MySQL commands into their inputs that can affect the underlying database and inadvertently expose your private data.
Protecting Against SQL Injection

- Use the `mysqli_real_escape_string` on ALL database inputs.
- E.G.

```
$sql = "SELECT * FROM x WHERE y ="". mysqli_real_escape_string($z)."";";
```
MySQL Architecture (Simplified)
SQLite Architecture Simplified

Application

SQLite Library

Local I/O

File System
# Mongo The Basics By Analogy

<table>
<thead>
<tr>
<th>MySQL</th>
<th>Mongo</th>
</tr>
</thead>
<tbody>
<tr>
<td>database</td>
<td>database</td>
</tr>
<tr>
<td>table</td>
<td>collection</td>
</tr>
<tr>
<td>row</td>
<td>document</td>
</tr>
<tr>
<td>column</td>
<td>field</td>
</tr>
<tr>
<td>primary key</td>
<td>primary key</td>
</tr>
</tbody>
</table>
What is a Document?

- A document is a collection of key-value pairs that looks an awful lot like JSON.
- JSON? - Let's review.
JSON - Remember Me?

```json
{
  "numeric_key" : 9,
  "string_key" : "Hello",
  "bool_true_key" : true,
  "bool_false_key" : false,
  "null_key" : null,
  "array_key" : [true, "2", "three"],
  "dictionary_key" : {
    "a" : "there"
  }
}
```
Every document inserted into a collection is assigned a unique _id field if no such field is specified.

_id is generated based on a number of factors (time, previous inserts, current machine).
MongoDB - Quick Review

- Use a database
  - use database_name
- Insert record
  - db.collection.insert({<document>});
- Show all databases
  - show dbs
- Show all collections
  - show collections
Finding

- **Basic**
  - `db.collection.find()`

- **Exact Match**
  - `db.collection.find(\{"k1" : "v1"\})`

- **Comparison:**
  - `db.collection.find(\{"k1" : {$lt : "v1"}\})`
  - `$lt, $lte, $gt, $gte, $ne`

- **Negation**
  - `db.collection.find(\{"k1" : {$not : {expr}}\})`

- **Membership**
  - `db.collection.find(\{"k1" : {$in : ["v1","v2",...,"vn"]}\})`

- **Existence Testing**
  - `db.collection.find("k1" : {$exists : true})`
More Finding

- With Limit
  - `db.collection.find().limit(n)`
- With Sorting
  - `db.collection.find().sort({"key_1" : 1|-1, "key_2" : 1|-1})`
- Count
  - `db.collection.find().count()`
Being Distinct

- Get all unique values for specific key in collection.
  - `db.collection.distinct("k1")`

- Get number of distinct values
  - `db.collection.distinct("k1").length`
  - This is different syntax than count
Projecting

- Return all fields
  - `db.collection.find({},{});`

- Return only specified field (and _id)
  - `db.collection.find({},{"k1" : 1});`

- Return all but specific field
  - `db.collection.find({},{"k1" : 0})`

- Return only specific fields (without _id)
  - `db.collection.find({},{"k1" : 1, "_id" : 0});`
Finally - The Final

- Areas we will cover
  - Relational Design
  - MySQL
  - PHP
  - PHP + MySQL
  - SQLite
  - MongoDB