Data Management
USI
12/5/2015
Server-Side Scripting (PHP)
Outline

• HTML Basics
• PHP Basics
• PHP Data and Control
• Programming in the large
• Databases
• Stateful Web Applications
About HTML

- HyperText Markup Language
  - HyperText = contains clickable links
  - Markup Language = structuring and rendering instructions

- Used for most of the internet
  - Static HTML = simple file retrieved from server’s disk, or generated by server-side script, e.g., PHP
  - Dynamic HTML = HTML containing client-side script

![Diagram of the HTTP request process between User, Browser, and Server]

Data Management USI 12/5/15
Related Languages

- Markup languages: troff, LaTeX, texinfo
- 1991, Tim Berners-Lee published description of HTML as application of SGML
- HTTP = HyperText Transfer Protocol
- Standards vs. Browser implementations
- 1996, XML = eXtensible Markup Language
  - Simpler variant of SGML
  - Used to define markup languages, such as HTML
- 2000, XHTML = HTML that conforms to XML
How to Write + Run Code

• Create web site:

  ssh euler.inf.unisi.ch
  ssh energon1
  mkdir $HOME/public_html $HOME/public_html/php
  chmod 701 $HOME $HOME/public_html $HOME/public_html/php
  cd $HOME/public_html/php
  echo '<h1>Welcome to ' `pwd` ' '</h1>' > index.html
  chmod 604 index.html

• Use a web browser to look at your web site:

  http://www.inf.usi.ch/~user_id/php/index.html
  http://www.inf.usi.ch/~user_id/php/
Structure of an HTML Document

Text Editor (e.g. Emacs)  Web Browser (e.g. Firefox)
Lexical Peculiarities

Comment: <!-- ... -->
Element: <table>...</table>
Start tag: <table>
Attribute: border=5
Contents: (between matching tags)
End tag: </table>
Empty element: <hr .../>
Entity: &
HTML Documentation

• [http://www.w3schools.com/html/](http://www.w3schools.com/html/)

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>html</td>
<td>document</td>
</tr>
<tr>
<td>head, title</td>
<td>header</td>
</tr>
<tr>
<td>body</td>
<td>main part</td>
</tr>
<tr>
<td>h1, h2, h3</td>
<td>headings</td>
</tr>
<tr>
<td>p</td>
<td>paragraph</td>
</tr>
<tr>
<td>br</td>
<td>line break</td>
</tr>
<tr>
<td>hr</td>
<td>horizontal line</td>
</tr>
<tr>
<td>pre</td>
<td>preformatted</td>
</tr>
<tr>
<td>b</td>
<td>bold</td>
</tr>
<tr>
<td>i</td>
<td>italic</td>
</tr>
<tr>
<td>a href</td>
<td>hyperlink</td>
</tr>
<tr>
<td>ul</td>
<td>• unordered list</td>
</tr>
<tr>
<td>ol</td>
<td>1. ordered list</td>
</tr>
<tr>
<td>li</td>
<td>list entry</td>
</tr>
<tr>
<td>table</td>
<td>table</td>
</tr>
<tr>
<td>tr</td>
<td>table row</td>
</tr>
<tr>
<td>td</td>
<td>table data</td>
</tr>
<tr>
<td>th</td>
<td>table head</td>
</tr>
</tbody>
</table>
Input and Output

```html
<html>
<head>
    <title>Form test</title>
</head>
<body>
<form name="frmTest" action="script_url.php" method="get"> <!-- or: method="post" -->
    Text field: <input type="text" name="txt">
    <br/>
    Radio buttons:
    A <input type="radio" name="rdo" value="a"/>
    B <input type="radio" name="rdo" value="b"/>
    <br/>
    Check boxes:
    C <input type="checkbox" name="chk" value="c"/>
    D <input type="checkbox" name="chk" value="d"/>
    <br/>
    Submit button:
    <input type="submit" value="submit"/>
</form>
</body>
</html>
```
HyperText Transfer Protocol

- HTTP GET: parameters encoded in URL
- HTTP POST: parameters in message header
Outline

• HTML Basics
• PHP Basics
• PHP Data and Control
• Programming in the large
• Databases
• Stateful Web Applications
About PHP

• PHP: Hypertext Processor
  – Recursive acronym
  – HTML = HyperText Markup Language

• Claim to fame: simplicity
  – Script embedded in HTML generates HTML
  – Libraries for MySQL, Oracle, PDF, XML

User

Browser

Server

click or type URL

HTTP request

Run PHP to create HTML

render markup

serve HTML
## Related Languages

<table>
<thead>
<tr>
<th>Date</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 1995:</td>
<td>PHP = Personal Home Page Tools, Rasmus Lerdorf, CGI scripts written in C</td>
</tr>
<tr>
<td>April 1996:</td>
<td>PHP/FI scripting language, Rasmus Lerdorf</td>
</tr>
<tr>
<td>June 1998:</td>
<td>Engine rewritten by Zeev Suraski and Andy Gutsmans in Tel Aviv</td>
</tr>
<tr>
<td>May 2000:</td>
<td>Parser rewritten again: “Zend Engine”</td>
</tr>
<tr>
<td>May 2004:</td>
<td>PHP 5, current language version</td>
</tr>
</tbody>
</table>

- Evolution from script collection to scripting language
- Other server-side languages embedded in HTML: VisualBasic (ASP), Java (JSP)
Secure Your Website!

• Put the following in $HOME/public_html/php/.htaccess:
  AuthType Basic
  AuthUserFile /home/user_id/.htpasswd
  AuthName "Members ONLY"
  require valid-user

• Create a user name and password:
  /usr/bin/htpasswd -c $HOME/.htpasswd user_name
  # make up a new password for user_name, don't forget it
  chmod 604 $HOME/.htpasswd $HOME/public_html/php/.htaccess

• Use a web browser, that will request authorization:
  http://www.inf.usi.ch/~user_id/php/index.html
How to Write + Run Code

• See: https://discussions.apple.com/docs/DOC-3083
• Put the following in $HOME/public_html/php/hello.php:
  ```php
  <html><body>
  <?php
  if(!empty($_GET['who'])) { echo "Hi, {$_GET['who']}."; }
  ?></body></html>
  ```
  ```php
  form action="<?php echo $_SERVER[ PHP_SELF]; ?>"
  method=get>
  Who shall be greeted: <input type="text" name="who" />
  </form>
  </body></html>

• Set the permissions:
  ```sh
  chmod 604 $HOME/public_html/php/hello.php
  ```
• Use a web browser to look at your php script:
  ```sh
  http://www.inf.usi.ch/~user_id/php/hello.php
  ```
Embedding Code in Web Pages

- Browser doesn’t see PHP script, only result of running script on server
- PHP code gets replaced by its own output (printed by “echo” and other functions)
- Four styles of embedding PHP:
  - XML style (preferred!)  `<?php ... ?>`
  - SGML style  `<? ... ?>`
  - ASP style  `<% ... %>`
  - Script style (deprecated)
    ```html
    <script language="php">
    ... </script>
    ```
- Variant of SGML style:  `<?= ... ?>` is shorthand for  `<? echo ... ?>`
Self-Processing Pages

- **User**
  - Click or type URL
  - Enter form data, submit
  - Render markup

- **Browser**
  - HTTP GET
  - HTTP GET or POST
  - Serve HTML

- **Server**
  - Run PHP, which creates HTML with form
  - Serve HTML
  - Run PHP, which creates HTML with result

• Same PHP script, different HTML response
**Input and Output**

- **Input:** EGPCS superglobals
  - `$_ENV`, `$_GET`, `$_POST`, `$_COOKIE`, `$_SERVER`
  - `$_REQUEST`: union of G,P,C
  - `$_FILES`: contains uploaded files
  - `$_SESSION`: persistent state across loads
- **Output:** printed text in HTML document
  - `echo`, `print()`, `printf()`
  - `var_dump()`, `print_r()`: print human-readable form for debugging; warning: problems with cycles
  - `phpinfo()`: prints lots of diagnostic information
Lexical Peculiarities

- Embedded in HTML with `<?php ... ?>`
- Variables are case sensitive; classes, functions, and keywords are case insensitive
- All variables (including arrays) begin with dollar sign ($), can be interpolated in string
- Semicolon required even after last statement in block, optional only before `?>`
- Single-line comments, or up to `?>`: #, //
- Multi-line comments, even across `?>`: /*...*/
- Literals: "s", 's', multi-line string, true, null
- Heredocs: continues after closing tag
PHP

Types

- scalar
  - (numeric)
    - int
    - float
  - string
  - bool
- collection
  - array
    - (associative)
      - subclass
  - object
- special
  - resource
  - NULL
### Type Conversions

<table>
<thead>
<tr>
<th>Value</th>
<th>bool</th>
<th>number</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bool</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>false</td>
<td>Identity</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>true</td>
<td>Identity</td>
<td>1</td>
</tr>
<tr>
<td><strong>number</strong></td>
<td>0</td>
<td>false</td>
<td>Identity</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>true</td>
<td>Identity</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td><strong>string</strong></td>
<td>&quot;&quot;</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>&quot;0&quot;</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>true</td>
<td>Numeric prefix</td>
</tr>
<tr>
<td><strong>null</strong></td>
<td>false</td>
<td>0</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td><strong>array</strong></td>
<td>empty</td>
<td>false</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td><strong>object</strong></td>
<td>empty</td>
<td>false</td>
<td>Error</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>
# Variable Declarations

<table>
<thead>
<tr>
<th>Implicit</th>
<th>echo $a + 1; $b = 5;</th>
<th>Read NULL if non-existent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant (global)</td>
<td>define('PI', &quot;3.14&quot;) echo PI</td>
<td>Not a variable, don’t need dollar sign ($)</td>
</tr>
<tr>
<td>Global, used locally</td>
<td>global $g; $g = $g + 1;</td>
<td>Otherwise, write creates new local $g</td>
</tr>
<tr>
<td>Local, unlimited lifetime</td>
<td>static $s; $s++;</td>
<td>Otherwise, value forgotten after return</td>
</tr>
</tbody>
</table>
## Operators

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new</code></td>
<td>N</td>
<td>Create object</td>
</tr>
<tr>
<td><code>[…]</code></td>
<td>L</td>
<td>Array subscript</td>
</tr>
<tr>
<td><code>++</code>, <code>--</code></td>
<td>1 N</td>
<td>Auto-increment / decrement</td>
</tr>
<tr>
<td><code>~</code>, <code>@</code>, <code>(int)</code>, <code>(array)</code>, ...</td>
<td>1 N</td>
<td><code>~</code>: bitwise negation; <code>@</code>: inhibit errors; <code>(type)</code>: cast</td>
</tr>
<tr>
<td><code>instanceof</code></td>
<td>2 N</td>
<td>Class/interface membership</td>
</tr>
<tr>
<td><code>!</code></td>
<td>1 R</td>
<td>Logical negation</td>
</tr>
<tr>
<td><code>*</code>, <code>/</code>, <code>%</code></td>
<td>2 L</td>
<td>Multiplicative</td>
</tr>
<tr>
<td><code>+</code>, <code>-</code>, <code>. </code></td>
<td>2 L</td>
<td>Additive; <code>. </code>: string concatenation</td>
</tr>
<tr>
<td><code>&lt;&lt;</code>, <code>&gt;&gt;</code></td>
<td>2 L</td>
<td>Bitwise shift</td>
</tr>
<tr>
<td><code>&lt;</code>, <code>&lt;=</code>, <code>&gt;</code>, <code>&gt;=</code></td>
<td>2 N</td>
<td>Comparison</td>
</tr>
<tr>
<td><code>==</code>, <code>!=</code>, <code>&lt;</code>, <code>===</code>, <code>!==</code></td>
<td>2 N</td>
<td>Identity; <code>===</code>, <code>!==</code>: equal value + type</td>
</tr>
<tr>
<td><code>&amp;</code>, <code>^</code>, `</td>
<td>`</td>
<td>2 L</td>
</tr>
<tr>
<td><code>&amp;&amp;</code>, `</td>
<td></td>
<td>`</td>
</tr>
<tr>
<td><code>?:</code></td>
<td>3 L</td>
<td>Conditional</td>
</tr>
<tr>
<td><code>=</code>, <code>+=</code>, <code>-=</code>, ...</td>
<td>2 L</td>
<td>Assignment</td>
</tr>
<tr>
<td><code>and</code>, <code>or</code>, <code>xor</code></td>
<td>2 L</td>
<td>Logical (not all same precedence)</td>
</tr>
<tr>
<td><code>:</code></td>
<td></td>
<td>List separator</td>
</tr>
</tbody>
</table>
Outline

- HTML Basics
- PHP Basics
- PHP Data and Control
- Programming in the large
- Databases
- Stateful Web Applications
Finding PHP Mistakes

• If script encounters error on server, the browser just gets empty HTML ⇒ not helpful!

• For compile errors: run at command line
  - `php -f file` Parse and execute file
  - `php -s -f file` Syntax highlighted source
  - `php -l -f file` Lint (check syntax without running)

• For logic errors:
  - Use `echo` statements to see what gets executed
  - Use `var_dump()` calls to inspect data structures

• To view source in browser:
  `ln -s script.php script.phps`
Arrays

- Creation: 
  ```php
  $a = array(1, 2, 3);
  $b = array('cat' => 'meow', 'dog' => 'woof')
  ```

- Indexing: e.g., 
  ```
  $a[2], $b['dog'], $b['cat']
  ```
  - '3' and "3" and 3 are the same key
  - -3 is the same key as "-3"
  - Quotes around string keys are optional, and must be omitted for interpolation
  - Write to non-existent index inserts, e.g., `$$a[3]=4;`
  - Write without index appends, e.g., `$$a[]=4;`

- Remove: `unset($a[2])`, delete `unset($a)`

- Multiple assign: `list($x, $y) = array(1, 2, 3);`
Array Library Functions


- `range(low, high, [, step])` - Create array
- `count(var [, mode])` - Size
- `array_keys(input [, ...])` - Indices
- `array_values(input)` - Contents
- `array_key_exists(key, array)` - Contains
- `sort(array [, sort_flags])` - By values
- `rsort(array [, sort_flags])` - Descending
- `usort(array, cmp_function)` - By user function
- `array_push(array, var [, ...])` - Add to end
- `array_pop(array)` - Remove from end
- `array_walk(array, func [, data])` - Mapping
# Control Statements

<table>
<thead>
<tr>
<th>Conditional</th>
<th>if (expr) ... elseif (expr) ... else ... switch(expr){case expr:... ... default: ...}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed iteration loops</td>
<td>foreach ($arr as $val) ... foreach ($arrs $key =&gt; $val) ...</td>
</tr>
<tr>
<td>Indefinite loops</td>
<td>for (expr; expr; expr) ... while (expr) ... do ... while (expr);</td>
</tr>
<tr>
<td>Unstructured control</td>
<td>break [expr]; # expr = loop levels to skip continue [expr]; # expr = loop levels to skip exit [expr]; # expr = error message return [expr]; # expr = return value</td>
</tr>
<tr>
<td>Directive</td>
<td>declare (directive) ... # rarely used</td>
</tr>
</tbody>
</table>
Alternative Control Syntax

if ($x < $y):
    echo "then branch";
    $min = $x;
else:
    echo "else branch";
    $min = $y;
endif;

if ($x < $y) {
    echo "then branch";
    $min = $x;
} else {
    echo "else branch";
    $min = $y;
}

Also available for other control statements, e.g.:

- while(expr): ... endwhile;
- for(expr; expr; expr): ... endfor;
- switch(expr):case expr:... ... default:... endswitch;
References

“Variable variable”
- Store name of one variable as string in other variable
- Also known as “soft reference”

\$x = 123;
\$r = 'x';
echo $$r;
\$r = '';

“Alias”
- Make two variables refer to the same memory location
- Also known as “hard reference”

\$x = 123;
\$r = &\$x;
echo $r;
unset $r;
Writing Subroutines

• Declaration: `function [&]id (arg*) { ... }`
  – To return a value: `return expr;`
  – &: return alias for result (hard reference)

• Arguments: `arg ::= [&]$id [ = expr]`
  – Call-by-value, even for arrays
  – &: call-by-reference
  – `[ = expr]`: optional parameter, default value
  – Empty `(arg*)`: `$my_array = func_get_args()`

• Variable functions: same as variable variables
Anonymous Functions

• Creating new variable function from strings:
  
  \$x = \text{create function}(\text{args}, \text{code});

• Example script:

```php
<?php

function callFn($fn, $x) { $fn($x); }
function printIt($it) { print "printIt $it\n"; }

# pass reference to named subroutine; prints "printIt Hello"
callFn("printIt", "Hello");
# pass reference to anonymous subroutine; prints "lambda Hi"
callFn(create_function('$it', 'print "lambda $it\n";'), "Hi");

?>
```

• Useful for call-backs, e.g.,
  `usort(array, cmp_function)`
Outline

- HTML Basics
- PHP Basics
- PHP Data and Control
- Programming in the large
- Databases
- Stateful Web Applications
Structure of a PHP Application

• Literal inclusion of code from file:
  
  require 'fileName' ; fatal if non-existent
  
  include 'fileName' ; warn if non-existent
  
  require_once 'fileName' ; no effect if repeated
  
  include_once 'fileName' ; no effect if repeated

• Use @include to suppress the warning

• Convention: filename extension .inc

• No separate scope / namespace for included code, may cause proliferation of globals
# Using Objects

```php
require_once 'Apple.inc';

$a1 = new Apple(150, "green");
$a2 = new Apple(150, "green");

$a2->color = "red";
$varvar = "weight";
$a2->$varvar = 220;

echo $a1->prepare("slice") . "<br/>
";
echo $a2->prepare("squeeze") . "<br/>
";
```

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>require_once 'Apple.inc';</code></td>
<td>Include file</td>
</tr>
<tr>
<td><code>$a1 = new Apple(150, &quot;green&quot;);</code></td>
<td>Constructor calls</td>
</tr>
<tr>
<td><code>$a2 = new Apple(150, &quot;green&quot;);</code></td>
<td>Constructor calls</td>
</tr>
<tr>
<td><code>$a2-&gt;color = &quot;red&quot;;</code></td>
<td>Property access</td>
</tr>
<tr>
<td><code>$varvar = &quot;weight&quot;;</code></td>
<td>Property access</td>
</tr>
<tr>
<td><code>$a2-&gt;$varvar = 220;</code></td>
<td>Property access</td>
</tr>
</tbody>
</table>
| `echo $a1->prepare("slice") . "<br/>
";` | Method calls |
| `echo $a2->prepare("squeeze") . "<br/>
";` | Method calls |
class Fruit {
    var $weight = 0;
    function __construct($weight) {
        $this->weight = $weight;
    }
    function pluck() {
        return "fruit(" . $this->weight . "g)";
    }
    function prepare($how) {
        return $how . "d " . $this->pluck();
    }
}
Inheritance in PHP

class Fruit {
    var $weight = 0;
    function __construct($weight) {
        $this->weight = $weight;
    }
    function pluck() {
        return "fruit(" . $this->weight . "g)";
    }
    function prepare($how) {
        return $how . "d " . $this->pluck();
    }
}

class Apple extends Fruit {
    var $color = "green";
    function __construct($weight, $color) {
        $this->weight = $weight;
        $this->color = $color;
    }
    function pluck() {
        return $this->color . " apple";
    }
}
More on Classes

• Modifiers: method (abstract, final), property (public, private, protected, const), both (static)
• Static member access: ::, self::, parent::
• Interface: like in Java, all methods implicitly abstract; class can extend class and implement interfaces
• Access to non-existent property turns into method call: __get($propName) or __set($propName, $value)
• Called on object death: __destruct()
• Introspection: class_exists, get_declared_classes, get_class_methods, get_class_vars, get_parent_class, is_object, get_class, method_exists, get_object_vars
Scopes and Visibility

• Locals: scope is entire function, not just block
• Globals:
  – “global $x;” is shorthand for “$x = &$GLOBALS['x'];”
  – register_globals in php.ini causes EGPCS to be spilled into globals; that’s bad for security
• Nested functions:
  – Inner function does not see outer locals/arguments
  – Inner function globally visible after first call to outer
• Modules: require/include don’t affect scoping
• Classes: public, private, protected properties
PHP Documentation

- CIMS web scripting instructions: [http://www.cims.nyu.edu/systems/userservices/webhosting/](http://www.cims.nyu.edu/systems/userservices/webhosting/)
- Tutorial: [http://www.w3schools.com/](http://www.w3schools.com/)
Evaluating PHP

**Strengths**
- Simplicity
- Portability
- Large libraries
- Many database bindings
- Popularity

**Weaknesses**
- Error handling
- Lack of scalability
  - Compared to Java
- Low-level
  - Compared to Ruby on Rails or Google Web Toolkit
Outline

- HTML Basics
- PHP Basics
- PHP Data and Control
- Programming in the large
- Databases
- Stateful Web Applications
Why use a Database?

• Store data for months or years
  – Database may live longer than the web application(s) that you write for it

• Why not just use simple ad-hoc files?
  – Database remains consistent in the presence of multiple concurrent accesses
  – Database scales better when there is a lot of data, or a lot of accesses
  – Don’t reinvent the wheel
Relational Databases

- Database = collection of tables
- Table = relation = set of rows w/ same columns
- Row = tuple = one value per column
- Column = attribute = name+primitive type
- Only store primitive values, never nest tables

<table>
<thead>
<tr>
<th>Recipe</th>
<th>Ingredient</th>
<th>RecCui</th>
<th>RecIng</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
<td>rec</td>
<td>rec</td>
</tr>
<tr>
<td>cake</td>
<td>flour</td>
<td></td>
<td>cake</td>
</tr>
<tr>
<td>gravy</td>
<td>salt</td>
<td></td>
<td>cake</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>serves</th>
<th>name</th>
<th>taste</th>
</tr>
</thead>
<tbody>
<tr>
<td>cake</td>
<td>4</td>
<td>flour</td>
<td>bland</td>
</tr>
<tr>
<td>gravy</td>
<td>4</td>
<td>salt</td>
<td>salty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>continent</th>
<th>rec</th>
<th>cui</th>
<th>ing</th>
<th>qty</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italian</td>
<td>Europe</td>
<td></td>
<td></td>
<td>cake</td>
<td>2.5</td>
<td>cup</td>
</tr>
<tr>
<td>Indian</td>
<td>Asia</td>
<td></td>
<td></td>
<td>cake</td>
<td>3</td>
<td>tbsp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>taste</th>
<th>rec</th>
<th>cui</th>
<th>ing</th>
<th>qty</th>
<th>unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>gravy</td>
<td>salt</td>
<td></td>
<td></td>
<td>sugar</td>
<td>10</td>
<td>g</td>
</tr>
<tr>
<td>gravy</td>
<td>sugar</td>
<td></td>
<td></td>
<td>cake</td>
<td>2</td>
<td>tsp</td>
</tr>
</tbody>
</table>

Data Management USI 12/5/15
UML with Associations

**Concepts**

**Association**
(binary relationship)

**Constraint on number of objects**

- **Recipe**
  - name PK
  - serves

- **Cuisine**
  - name PK
  - continent

- **Ingredient**
  - name PK
  - taste

- **Amount**
  - qty
  - unit

- **RecIng**

- **PK = primary key**

**Association-class**

Data Management USI 12/5/15
From UML to Relations

• For each class, create a relation
  • Relation name = class name
  • Relation columns = class attributes
• For each association, create a relation
  • Relation name = combine class names
  • Relation columns = primary keys of both classes, plus attributes of association class, if any
Most database products have additional primitive types.
About SQL

• **Structured Query Language**
  – Query information from relational database
  – Declarative: describe what information to find, not how to find it

• **SQL consists of two parts**
  – DDL = Data Definition Language
  – DML = Data Manipulation Language

• **Each database product (sqlite, MySQL, Oracle, DB2, …) has own SQL dialect**
  – We use sqlite in this course
How to Write + Run Code

• From PHP script
  → later in today’s lecture

• By hand, from command line

energon1:~ /usr/bin/sqlite3 test.db
SQLite version 3.3.6
Enter ".help" for instructions
sqlite> CREATE TABLE RecIng(rec VARCHAR(50), ing VARCHAR(50), qty FLOAT, unit VARCHAR(10), PRIMARY KEY(rec, ing));
sqlite> INSERT INTO RecIng VALUES('cake','flour',2.5,'cup');
sqlite> INSERT INTO RecIng VALUES('cake','milk',3,'tbsp');
sqlite> INSERT INTO RecIng VALUES('gravy','salt',2,'tsp');
sqlite> INSERT INTO RecIng VALUES('gravy','sugar',10,'g');
sqlite> SELECT ing, qty FROM RecIng WHERE rec = 'cake';
flour|2.5
milk|3.0
sqlite>
Create Table Statement

CREATE TABLE RecIng(rec VARCHAR(50), ing VARCHAR(50), qty FLOAT, unit VARCHAR(10), PRIMARY KEY(rec, ing));

createTable ::= CREATE [TEMP|TEMPORARY] TABLE (newTable | derivedTable)
newTable ::= [IF NOT EXISTS] [id .] id (column* (, tableConstraint)* )
derivedTable ::= [id .] id AS select
column ::= id [type] ([CONSTRAINT id] columnConstraint)*
columnConstraint ::= NOT NULL [conflict] | UNIQUE [conflict]
| PRIMARY KEY [sortOrder] [conflict] [AUTOINCREMENT]
| CHECK (expr) | DEFAULT expr | COLLATE collationName
tableConstraint ::= PRIMARY KEY (id*) [conflict]
| UNIQUE (id*) [conflict] | CHECK (expr)
conflict ::= ON CONFLICT conflictAlgorithm
conflictAlgorithm ::= ROLLBACK|ABORT|FAIL|IGNORE|REPLACE
**Insert Statement**

```sql
INSERT INTO RecIng VALUES('cake','flour',2.5,'cup');
INSERT INTO RecIng VALUES('cake','milk',3,'tbsp');
INSERT INTO RecIng VALUES('gravy','salt',2,'tsp');
```

```sql
insert ::= 
    INSERT [OR conflictAlgorithm] INTO [id .] id [(id*)] VALUES (expr*)
```

```sql
insert ::= 
    INSERT [OR conflictAlgorithm] INTO [id .] id [(id*)] select
```

```sql
conflictAlgorithm ::= ROLLBACK | ABORT | FAIL | IGNORE | REPLACE
```
Select Statement

```
SELECT ing, qty
FROM RecIng
WHERE rec = 'cake';
```
List of sqlite Statements

Data Definition Language (DDL)

- (CREATE | ALTER | DROP) TABLE
- (CREATE | DROP) INDEX
- (CREATE | DROP) TRIGGER
- (CREATE | DROP) VIEW
- CREATE VIRTUAL TABLE
- ATTACH DATABASE
- DETACH DATABASE
- ANALYZE
- REINDEX
- VACUUM

Data Manipulation Language (DML)

- INSERT
- SELECT
- UPDATE
- REPLACE
- DELETE
- BEGIN TRANSACTION
- COMMIT TRANSACTION
- ROLLBACK TRANSACTION
- END TRANSACTION
- EXPLAIN
- PRAGMA
Lexical Peculiarities

• Case insensitive
• Commands end with semicolon (;)
• Single-line comments: --...
• Multi-line comment: /*...*/
• String literal: 's'
  – Escape single quote (') in string with another single quote (''), not with backslash (\')
• Identifier: simple id or quoted "id"
  – Quoted identifier can contain any character
  – Quoted identifier can be same as keyword
## Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-</code>, <code>+</code>, <code>~</code>, <code>NOT</code></td>
<td>Negation</td>
</tr>
<tr>
<td>CASE <code>[expr] (WHEN expr THEN expr)</code>+ <code>[ELSE expr] END</code></td>
<td>Conditional</td>
</tr>
<tr>
<td>CAST <code>(expr AS type)</code></td>
<td>Conversion</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>*</code>, <code>/</code>, <code>%</code></td>
<td>Multiplicative</td>
</tr>
<tr>
<td><code>+</code>, <code>-</code></td>
<td>Additive</td>
</tr>
<tr>
<td><code>&lt;&lt;</code>, <code>&gt;&gt;</code>, <code>&amp;</code>, `</td>
<td>`</td>
</tr>
<tr>
<td><code>&lt;</code>, <code>&lt;=</code>, <code>&gt;</code>, <code>&gt;=</code></td>
<td>Comparison</td>
</tr>
<tr>
<td><code>=</code>, <code>==</code>, <code>!=</code>, <code>&lt;</code>, <code>IN</code></td>
<td>Identity</td>
</tr>
<tr>
<td>`expr [NOT](LIKE</td>
<td>GLOB</td>
</tr>
<tr>
<td><code>expr [NOT] BETWEEN expr AND expr</code></td>
<td>Comparison</td>
</tr>
<tr>
<td><code>AND</code>, <code>OR</code></td>
<td>Logic</td>
</tr>
<tr>
<td><code>[EXISTS] (select)</code></td>
<td>Query-in-expr</td>
</tr>
</tbody>
</table>
Library Functions

- **Aggregate**: avg, count, group_concat, max, min, sum, total
- **String**: glob, length, like, lower, ltrim, quote, replace, rtrim, soundex, substr, trim, upper
- **Number**: abs, max, min, random, round
- **Misc**: coalesce, ifnull, hex, last_insert_rowid, load_extension, nullif, randomblob, sqlite_version, typeof, zeroblob
- **Date+time**: date, time, datetime, julianday, strftime
SQL Documentation

- Read a standard databases text book
- sqlite: http://www.sqlite.org
- MySQL: http://www.mysql.com
- Tutorial: http://www.w3schools.com/sql/default.asp
Outline

• HTML Basics
• PHP Basics
• PHP Data and Control
• Programming in the large
• Databases
• Stateful Web Applications
Scope in Server-Side Scripts

- Request
  - $_GET, $_POST
  - global $g;
- Session
  -)$_SESSION
- Application
  - $_COOKIE
database
Example, Revisited

Change `<..dir..>` to your home directory;
create data directory; chmod a+rwx $HOME/data

```php
<html><body><?php
class MyDB extends SQLite3 {
    function __construct() {
        $this->open('./data/sqlite3'); }
}$d = new MyDB();
$rows = $d->query("select * from T");
if (!$rows) {
    echo "table does not yet exist, creating it ...<br>";
    $q = "create table T(I integer, S char(10))";
    if (!$d->query("create table T(I integer, S char(10))")) {
        die($err);
    }
    $d->query("insert into T values(0, 'n')");
}
$results = $d->query("select I from T where S='n'");
$row = $results->fetchArray();
echo "T[S=n][I]=" . $row['I'] . "; reload for ++<br>";
$d->query("update T set I = I+1 where S='n'");
echo "delete data/sqlite3 to start over<br>
??></body></html>
```
PHP Bindings for Database

• See also: [http://www.php.net/sqlite](http://www.php.net/sqlite)
  - `void sqlite3::open(string $file [,..])`
  - `SQLite3Result sqlite3::query(string $query)`
  - `array SQLite3Result::fetchArray([int $mode])`
  - `bool sqlite3::close(void)`
  ... and many more functions for PHP+sqlite

• PHP also has many other database bindings
HTTP Cookies

Browser: store cookie on disk, return with next request

HTTP request

HTTP response

HTTP request

PHP script: setcookie('id', 123)

PHP script: read $_COOKIES['id']
Session Variables

Browser:
store ID on disk, or not

HTTP request
Generate random session ID
If browser accepts cookies:
   Send ID as cookie in HTTP head
Else:
   Send ID in customized HTML

HTTP response
PHP script: write
   session_start()
   $_SESSION['x'] = 5

Server:
store mapping from ID to set of session variables

HTTP request
If browser accepts cookies:
   Return cookie in HTTP header
Else:
   Rewritten URL or hidden form field

PHP script: read
   $_SESSION['x']
Cookies vs. Session Variables

<table>
<thead>
<tr>
<th>Cookies</th>
<th>Session variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored at client</td>
<td>Stored at server</td>
</tr>
<tr>
<td>May be rejected by browser</td>
<td>If browser rejects cookies, fall back on other mechanisms</td>
</tr>
<tr>
<td>More global scope</td>
<td>Local to server host</td>
</tr>
<tr>
<td>Longer lifetime</td>
<td>Deleted after session</td>
</tr>
</tbody>
</table>
Common PHP Mistakes

• Compiler errors (blank page), permissions errors (access denied)

• Must set cookie (or call `session_start()`) before generating any HTML, so cookie can go in HTTP header, before payload

• Security errors
  – SQL injection: using unvalidated user input as part of database query
  – Cross-site scripting: using unvalidated user input as part of public forum HTML output
Last Slide

• Today’s lecture
  – HTML, HTTP
  – Server-side scripting
  – PHP
  – Web applications
  – Databases