

# PHP and MySQL

INFO 344 Winter 2007

## Administrivia

- r Writing Assignment 1 due today
- r MVC Lab due today
  - λ 10 points
- r No lab from last time
- r Extra credit proposals by Feb 20
  - λ Email is fine
  - λ Presentations on Mar 8

## Project and Course Grading

- r Revised Deliverables for Project
  - λ Two additional diagramming assignments
    - r Model and View classes
    - r Database tables
  - λ Less emphasis on final product; more on process
  - λ No Web Services requirement
- r Revised Course Calendar
- r New [Grading Summary](#) page
  - r Full list of labs
  - r Final grade worksheet

## Today

- r Databases in PHP
- r Miscellaneous database issues
- r Database security
- r Miscellaneous PHP issues
- r Objects in databases

## A note on terminology

- r In formal database speak:
  - λ An *entity* is a general class of object (e.g. Car, Book, Purchase)
  - λ Entities have *attributes* (color, name, id)
  - λ A *tuple* is a specific instance of an entity (my Honda Civic, *The Grapes of Wrath*)
  
- λ table = entity
- λ column = attribute
- λ row = tuple
  
- r For our purposes, "rows" and "tables" are OK

## Review: PHP and MySQL

- r PHP provides a number of functions for interacting with MySQL databases
  - λ `mysql_connect()`
  - λ `mysql_db_query()`
  - λ `mysql_fetch_array()`

## mysql\_connect()

```
$connection = mysql_connect($hostname,  
$username, $password) or  
die($errorMsg);
```

- r Creates a connection to the DBMS with the specified parameters
- r Returns a DB connection object
- r Optional error message

```
$connection = mysql_connect("localhost",  
"jim", "mypassword") or die("Could not  
connect: " . mysql_error());
```

## mysql\_db\_query()

```
$result =  
mysql_db_query($databasename,  
$query, $connection);
```

- r Uses the specified connection and database name to execute the query
- r Returns a result object

```
/* gets all books in the library database  
and Books table */  
$result = mysql_db_query("library",  
"select * from Books", $connection);
```

## mysql\_fetch\_array()

```
$row = mysql_fetch_array($result);
```

- r Returns the next row from the result object as an associative array
- r Returns FALSE if there are no more rows

```
/* gets all books in the library database and  
Books table */  
$result = mysql_db_query("library", "select *  
from Books", $connection);  
$row = mysql_fetch_array($result);  
echo $row["title"]; // echo the first title attribute
```

## mysql\_num\_results()

```
$numrows = mysql_num_results($result);
```

- r Returns the number of rows in a query result
- r Only valid for SELECT statements

```
/* gets all books in the library database and  
Books table */  
$result = mysql_db_query("library", "select *  
from Books", $connection);  
$numrows = mysql_num_results($result);  
echo $numrows; // number of rows returned
```

## Example code

```
<?php  
// connect to the DBMS  
$connection = mysql_connect("localhost", "shaun",  
"mypassword");  
// select all books from the Library database  
$result = mysql_db_query("library", "select * from  
Books", $connection);  
// get the number of rows  
$numRows = mysql_num_rows($result);  
// loop through each book in the result set  
for ($i = 0; $i < $numRows; $i++)  
$row = mysql_fetch_array($result); // get the next  
row  
echo $row["title"]; // echo the title for each book  
}  
?>
```

## Using double quoted strings

- r In PHP, double quoted strings automatically parse variable names
- r Can use this to make our query code simpler

```
$user = "tiffany";  
$password = "bananas";  
$query1 = "SELECT * FROM Users WHERE  
username = \" . $user . \" AND password = \"  
 . $password . \"\";";  
$query2 = "SELECT * FROM Users WHERE  
username = '$user' AND password =  
'$password'";
```

## Database miscellany

- r Foreign keys revisited
- r Error handling
- r Working with dates and times
- r Database security
- r Uploading files

## Foreign key constraints

- r Foreign keys are used to reference one table entity from another
- r By default, MySQL does not enforce the integrity of foreign keys
  - λ "referential integrity"
- r However, we can use a special table type that supports additional features

## Foreign key constraints

```
CREATE TABLE `Author` (  
  id int auto_increment NOT NULL,  
  name varchar(300) NOT NULL,  
  PRIMARY KEY(id)  
) type=InnoDB;  
  
CREATE TABLE `Book` (  
  id int auto_increment not null,  
  title varchar(300) not null,  
  authorID int not null,  
  PRIMARY KEY(id),  
  FOREIGN KEY (authorID) REFERENCES Author(id)  
) type=InnoDB;
```

## Handling changes

```
CREATE TABLE `Book` (  
  id int auto_increment not null,  
  title varchar(300) not null,  
  authorID int not null,  
  PRIMARY KEY(id),  
  FOREIGN KEY (authorID) REFERENCES  
  Author(id)  
  ON UPDATE CASCADE  
  ON DELETE CASCADE  
) type=InnoDB;
```

## Mapping tables

- r Sometimes we have a many-to-many relationship in our data
  - λ e.g. multiple parents may have multiple children
- r One way to handle this is a table that consists only of foreign keys
  - λ Map children to parents
  - λ Parent: id, name
  - λ Child: id, name
  - λ ParentOf: parentID, childID

## Mapping Tables

Parents		
parentID	Name	Sex
1	John	M
2	Mary	F
3	Christine	F

Children		
childID	Name	Sex
1	Bobby	M
2	Sarah	F

ParentOf	
childID	parentID
1	1
1	2
2	1
2	3

## Mapping Tables

---

- r Who are Bobby's parents?  

```
SELECT Parents.* FROM Parents
INNER JOIN (Children INNER JOIN
ParentOf ON Children.childID =
ParentOf.childID)
ON Parents.parentID =
ParentOf.parentID
WHERE Children.Name)="Bobby";
```

## Handling errors

---

- r We can use `or die` and `mysql_error()` to provide error messages if our DB code fails
    - λ `or die`: print error msg if database action fails
    - λ `mysql_error()`: return database error
    - λ This works for all database functions
- ```
$results = mysql_db_query("library",
"select flavor from Books", $connection)
or die("Error! " . mysql_error());
```

## Dates and times

---

- r MySQL provides two mostly identical date/time types: **DATETIME** and **TIMESTAMP**
  - λ DATETIME is fine for most uses
  - λ TIMESTAMP features some additional updating features
- r Use `NOW()` to refer to the present time  

```
INSERT INTO Log(name, time)
VALUES ('Index updated', NOW());
```

## Date formats

---

- r PHP stores dates in the UNIX datetime format
  - λ integer value: seconds from January 1, 1970
- r MySQL stores dates in string format
- r Use PHP's `date()` function to convert PHP->MySQL
  - λ `$mysqlTime = date("Y-m-d G:i:s");`
- r Use PHP's `strtotime()` function to convert MySQL->PHP
  - λ `$phpTime = strtotime($row["date"]);`

---

## Database security

## Handling string values

---

- r When form inputs are passed to the database, we have two problems
  - λ Values with quotes may cause query to break ("Finnegan's Wake")
  - λ Bad people may use this flaw to manipulate the database – this is called an *SQL injection attack*

## SQL injection

- Imagine that we have a database query that checks a database to see if a username and password are valid

```
$username = $_REQUEST["username"];
$password = $_REQUEST["password"];
$query = "SELECT * FROM Users WHERE
username='$username' AND
password='$password'"
```

- If user and password are valid, should return one row
  - Otherwise, none will be returned

## SQL injection example

- A malicious user could enter in the password  
test' OR '1
- Query will then be  
SELECT \* FROM Users WHERE  
username='user' AND password='test' OR  
'1'
- This query will always return a result

## Patching the hole

- Use `mysql_real_escape_string()` to solve these problems

```
$username =
mysql_real_escape_string($_REQUEST[
"username"]);
$password =
mysql_real_escape_string($_REQUEST[
"password"]);
$query = "SELECT * FROM Users WHERE
username='$username' AND
password='$password'";
```

## Storing passwords

- It's better not to store user passwords in plain text
- Use MySQL's `MD5()` function to store password as an unrecognizable hash code
  - Plain text password can never be extracted from database

```
$username = $_REQUEST["username"];
$password = $_REQUEST["password"];
$query = "SELECT * FROM Users WHERE
username='$username' AND
password=MD5('$password')";
```

## File uploads

## File uploads

- We can store uploaded files in a database or in the file system
- For simplicity, we will upload files to an "uploads" directory and save their URL in the database
- Use HTML form's "file" input type to create a file upload window

## File upload process

---

1. HTML form passes file to PHP page
2. PHP automatically copies file to temporary directory
3. PHP script copies temp file to permanent location
4. PHP script adds new file's URL to database

## File upload form

---

```
<form name="upload"
action="upload.php"
enctype="multipart/form-data"
method="post">
  <input type="file"
name="uploadfile"/>
  <input type="submit"
value="Upload"/>
</form>
```

## File upload PHP

---

- r `$_FILES` array contains uploaded files
- r Provides file array with properties
  - λ `$_FILES['param']['name']`: original file name
  - λ `$_FILES['param']['tmp_name']`: path to temp file
- r Use `move_uploaded_file` to copy temporary file to a permanent location
  - λ Otherwise it will be deleted when script finishes

## Upload PHP example

---

```
echo $_FILES['uploadfile']['name']; //
  uploaded file name
echo
  $_FILES['uploadfile']['tmp_name'];
  // temp file name
move_uploaded_file($_FILES['uploadf
ile']['tmp_name'], 'uploads/' .
  $_FILES['uploadfile']['name']);
echo "File moved to " . 'uploads/' .
  $_FILES['uploadfile']['name'];
```

## User Logons

---

## pubcookie and PHP

---

- r Recall: `.htaccess` can be used to provide UW NetID authentication

```
AuthType UWNetID
AuthName "Application Name"
PubcookieAppID "Application Name"
require valid-user
```

## pubcookie and PHP

---

- r Using pubcookie, the user's logon name is available to PHP during their session

`$_ENV[REMOTE_USER]`

## pubcookie and PHP

---

- r Possibilities
  - λ Require NetID authN for you app
  - λ Use NetID as username in "user" table
  - λ Don't store (or worry about) passwords

## Data Access Objects

---

## Data access objects pattern

---

- r Data Access Objects are another *object-oriented design pattern*
- r Problem: we would like to load and store our program objects without worrying about SQL
  - λ Work at object level, not SQL
  - λ Make it somewhat easy to change our storage method
- r Solution: provide an abstraction layer between objects and data store

## General pattern

---

- r Provide a *Manager* class for passing objects to and from database
  - λ Book -> BookManager, User -> UserManager
- r Manager class provides methods for accessing database
  - λ Constructor makes DB connection
  - λ `getBook($id)`
  - λ `saveBook($book)`
  - λ `getBooksByAuthor($author)`

## Example: BookManager

---

```
<?php
class BookManager {
    var $connection;
    function BookManager() {
        $this->connect();
    }
    function addBook($book) { ... }
    function deleteBook($book) { ... }
    function getBook($id) { ... }
    function getAllBooks() { ... }

    function connect() {
        $this->connection = mysql_connect("localhost",
"root", "")
        or die("Could not connect: " .
mysql_error());
    }
}
?>
```

## Retrieving objects

```
function getBook($id) {
    // execute query
    $query = mysql_db_query("library", "select * from
    Library.Books where id=$id");
    // retrieve result
    $result = mysql_fetch_array($query);
    // pass to Book constructor
    $book = new Book($result["title"],
    $result["author"], $result["pages"], $result["isbn"]);
    $book->id = $result["id"];
    // return object
    return $book;
}
```

## Adding objects

```
function addBook($book) {
    mysql_db_query("library",
    "insert into Books (title, author,
    pages, isbn) values ('$book->title',
    '$book->author',
    $book->pages,
    '$book->isbn')");
    // return the id of the new book
    return mysql_insert_id();
}
```

## DAO in MVC

- We can consider Manager classes to be part of the model
  - Kept separate from View
  - Controller can create and store model objects

## Finally: default parameters in PHP

- Parameters in PHP can have default parameters

```
function add($a, $b = 1, $c = 2) {
    return $a + $b + $c;
}
...
echo add(3); // 6
echo add(3, 4); // 9
echo add(3, 4, 5); // 12
```