Lecture Overview

- 3.1 Student Topic 1 Presentations
- 3.2 Native Database Drivers – mysqli, sqlite
- 3.3 Database Abstraction and Parameters
- 3.4 PDO
- 3.5 Transactions
- Login Lab With Database

Server Side Languages

Web Design & Development

Day 3

Student Presentations

Topic 1

Database Access

Vendor Specific Database Extensions

Written to interface PHP directly with a specific type of database.

Typically written for fastest performance.

It is up to the application developer to implement security best practices to protect against attacks or anomalies (such as SQL injection).

Libraries for: CUBRID, DB++, dBase, filePro, Firebird/InterBase, FrontBase, IBM DB2, Informix, Ingres, MaxCB, Mongo, mSQL, Microsoft SQL, MySQL, Oracle OCI8, Ovrimos, Paradox, PostgreSQL, SQLite, Sybase, TokyoTyrant

http://us2.php.net/manual/en/refs.database.php

Database Access

Abstraction Layers

- Creates a layer between PHP and the underlying database technology.
- Makes it easier to adapt PHP application between different databases by changing connection strings.
- Access methods the same way regardless of DBMS. The abstraction layer “translates” into the underlying database technology using specific drivers.
- Most common is PDO (PHP Database Objects).
- Prepared statements have inherent protection against various potential attacks.

http://us2.php.net/manual/en/pdo.drivers.php

mysqli Native Driver

Why might you want to use a native driver? Or, why not?

These examples are being shown as a historical reference - normally as a practice you would prefer to use PDO for “most” database interaction.

```
Procedural Interface
$mysqli = mysqli_connect("example.com", "user", "password", "database");
$res = mysqli_query($mysqli, "SELECT 'Do not use ' AS _msg FROM DUAL");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];

Object-Oriented Interface
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: " . $mysqli->connect_error;
}
$res = $mysqli->query("SELECT 'choices.' AS _msg FROM DUAL");
$row = $res->fetch_assoc();
echo $row['_msg'];
```

mysqli Prepared Statements

```
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}
if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1), (2), (3)")) {
    echo "Table creation failed: (" . $mysqli->errno . ") " . $mysqli->error;
}
$res = $mysqli->query("SELECT id FROM test ORDER BY id ASC");
echo "Reverse order...\n";
for ($row_no = $res->num_rows - 1; $row_no >= 0; $row_no--) {
    $res->data_seek($row_no);
    $row = $res->fetch_assoc();
    echo " id = " . $row['id'] . "\n";
}
echo "Result set order...\n";
$res->data_seek(0);
while ($row = $res->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
```

PDO

- The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP.
- PDO provides a data-access abstraction layer, which means that, regardless of which database you're using, you use the same functions to issue queries and fetch data. PDO does not provide a database abstraction; it doesn't rewrite SQL or emulate missing features.
- A data-access abstraction layer makes it easier to connect to multiple types of databases regardless of their specific requirements - PDO is a “wedge” to provide unified access.

  - http://us1.php.net/manual/en/intro.pdo.php

PDO Connections

DSN: Data Source Name - connection string that tells PDO the type of database to access, where to access it, and other parameters.

```
$user=”root”;
$pass=”root”;
$dbh = new PDO('mysql:host=localhost;dbname=test;port=8889', $user, $pass);

// and now we're done; close it
$dbh = null;
```

Once $dbh is instantiated, we can the pass data to/from it.

When you’re finished with the database connection, release the object to free resources.

PDO Prepared Statements

- Besides making unified references to a database through PDO, it provides parameterization for increased security against things such as SQL injection.
- It also allows for making multiple SQL calls but with different values (for instance, inserting multiple rows at a time).
- Write your regular SQL statement, then replace the values with parameters. This can be done either as ordered positioned placeholders (using a “?” as a placeholder), or as named placeholders (using “:placeholdername”). It is recommended to use named placeholders.
- The prepare() method prepares a SQL statement for use by the PDO object.
- The bindParam() method substitutes in the placeholder values.
- The execute() method then runs the completed SQL statement against the datasource.

  - http://us1.php.net/manual/en/pdo.prepared-statements.php

PDO Query

PDO::query() executes an SQL statement in a single function call, returning the result set (if any) returned by the statement as a PDOStatement object.

```
 function getFruit($conn) {
     $sql = 'SELECT name, color, calories FROM fruit ORDER BY name';
     foreach ($conn->query($sql) as $row) {
         print $row['name'] . "\t";
         print $row['color'] . "\t";
         print $row['calories'] . "\n";
     }
}
```

The above example will output:

```
apple   red     150
banana  yellow  250
lemon   yellow  25
orange  orange  300
```

PDO Fetch Prepared Statements

Use placeholders in the SQL statement itself, in the example :calories

Bind the placeholder parameter to a variable, in the example $calories

Execute runs the SQL statement.

fetchAll retrieves the records as an key/value array and stores in $result

```
$calories = 150;
$color = 'red';
$sth = $dbh->prepare('SELECT name, color, calories
    FROM fruit
    WHERE calories < :calories AND color = :color');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindParam(':color', $color, PDO::PARAM_STR, 12);
$sth->execute();
$result = $sth->fetchAll();
print_r($result);
```

PDO Prepared Statement using Array

Parameters can passed as a single array at time of execution.

```
/* Execute a prepared statement by passing an array of insert values */
$calories = 150;
$color = 'red';
$sth = $dbh->prepare('SELECT name, color, calories
    FROM fruit
    WHERE calories < :calories AND color = :color');
$sth->execute(array(':calories' => $calories, ':color' => $color));
$result = $sth->fetchAll();
print_r($result);
```

Exercise 3.2: PDO Select Data

1. Add to your existing “fruits.php” script, and after the input form, echo out the fruits and colors that are listed in the table.
2. Insert a new record into the database - you should see the new row at the bottom of the page when you postback.

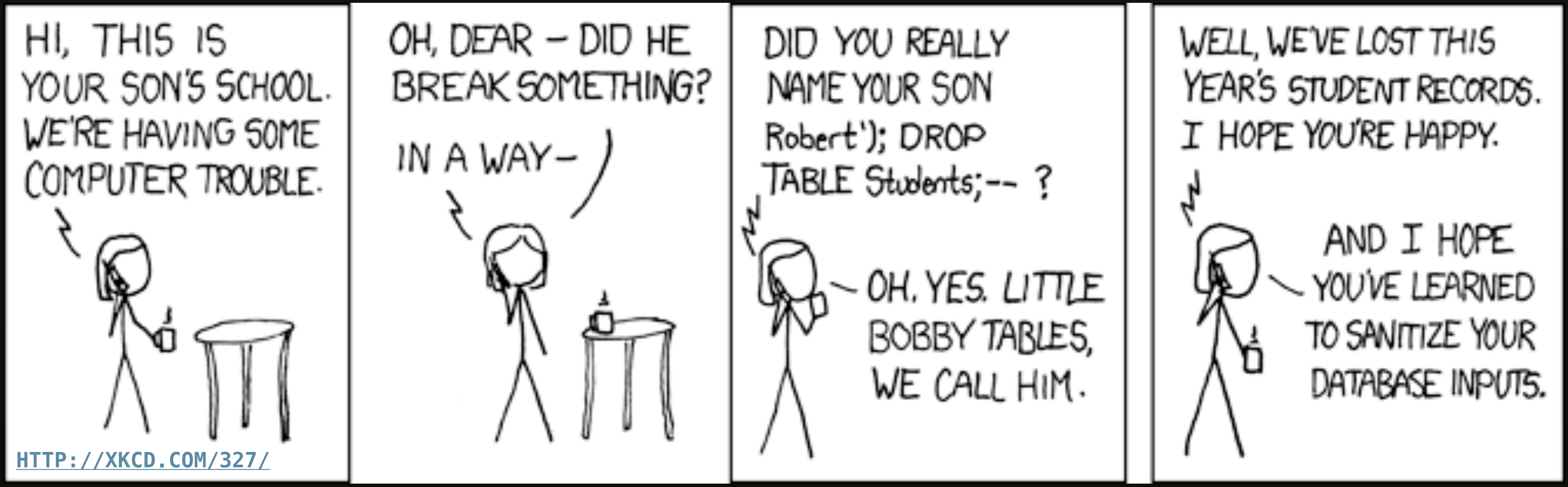
```php
$calories = 150;
$color = 'red';
$sth = $dbh->prepare('SELECT name, color, calories
    FROM fruit
    WHERE calories < :calories AND color = :color');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindParam(':color', $color, PDO::PARAM_STR, 12);
$sth->execute();
$result = $sth->fetchAll();
print_r($result);
```

How NOT to Use Databases

Substitute Data Directly in SQL

```
$name = ‘tangerine’;
$color = 'orange';
$sth = $dbh->prepare('UPDATE fruit SET color=’ . $color .
  ‘WHERE name = ‘ . $name);
$sth->execute();
```

mysql_real_escape_string


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.
HTTP://XKCD.COM/327/

Questions?

Lab 3: Databases

Refer to FSO Lab3 and Screencast3 for this assignment