Lecture 4 Overview

- Day3/Lab3 Questions
- Review Student Videos
- Lecture

- Database Connections

Database Access

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

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

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

http://us2.php.net/manual/en/book.mysqli.php

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These examples are being shown as a historical reference - normally as a practice you would prefer to use PDO for "most" database interaction.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 'Please, 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 to please everybody.' AS _msg FROM DUAL");
$row = $res->fetch_assoc();
echo $row['_msg'];
```

mysqli Prepared Statements

http://us2.php.net/manual/en/mysqli.quickstart.prepared-statements.php

```
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $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";
}
```

Database Access

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

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

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

- 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.

```
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insert one row
$name = 'one';
$value = 1;
$stmt->execute();

// insert another row with different values
$name = 'two';
$value = 2;
$stmt->execute();
```

Exercise: Insert Data into MySQL using PDO

- Create a table named "fruits" with an ID as your primary key, and columns for fruitname and fruitcolor. (Create a database named "ssl" if you have not already done so.)
- Create a PHP script named "fruits.php" with HTML form to input fruits and their color.
- Have the form post back to itself. Use a conditional to check whether $_POST has been set - if true, insert the data.
- Test inserting records into fruits, and verify that the rows have been successfully inserted by checking in Sequel Pro.

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

$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insert one row
$name = 'one';
$value = 1;
$stmt->execute();
```

PDO Query

http://us1.php.net/manual/en/pdo.query.php

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.

```
<?php
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
kiwi    brown   75
lemon   yellow  25
orange  orange  300
```

PDO Prepared Statements and Fetch

http://us1.php.net/manual/en/pdostatement.execute.php

http://us1.php.net/manual/en/pdostatement.fetchall.php

$result contains an array with the entire recordset returned.

Example #1 Execute a prepared statement with bound variables

```
<?php
/* Execute a prepared statement by binding PHP variables */
$calories = 150;
$colour = 'red';
$sth = $dbh->prepare('SELECT name, colour, calories
    FROM fruit
    WHERE calories < :calories AND colour = :colour');
$sth->bindParam(':calories', $calories, PDO::PARAM_INT);
$sth->bindParam(':colour', $colour, PDO::PARAM_STR, 12);
$sth->execute();
$result = $sth->fetchAll();
print_r($result);
?>
```

Example #2 Execute a prepared statement with an array of insert values (named parameters)

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

Exercise: Select Data from MySQL using PDO

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

SQLite

http://www.sqlite.org/

SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine.

https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/

```
try {
    /*** connect to SQLite database ***/
    $dbh = new PDO("sqlite:/path/to/database.sdb");
    }
catch(PDOException $e)
    {
    echo $e->getMessage();
    }
```

Exercise: Create SQLite Database and access using PDO

- Download the sqlite-manager plugin for Firefox.
- Create the same Fruits data schema you used in MySQL.
- Using your existing "fruits.php" script, alter your connection string and insert and read data using SQLite instead.
- Insert a new record into the database - you should see the new row at the bottom of the page when you postback.
- Verify that the inserted data exists in the SQLite database (using the manager).

```
try {
    /*** connect to SQLite database ***/
    $dbh = new PDO("sqlite:/path/to/database.sdb");
    }
catch(PDOException $e)
    {
    echo $e->getMessage();
    }
```

https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/

Today's Lab (Lab 4)

In this lab you will build upon Lab 3.

- Instead of saving the captured information from the form in session variables, write the user information to a users table in your database.
- Create a user login page where a user logs in using their stored username/pass. On successful login, set a session variable with their userID and Name from the database and display their avatar on the page along with their username.

SUBMIT: Source code zipped and uploaded on FSO Lab 4; Link to screencast video explaining code uploaded on FSO Video 4.