Accessing a Database from PHP

pluralsight
hardcore developer training

Lesson Objectives

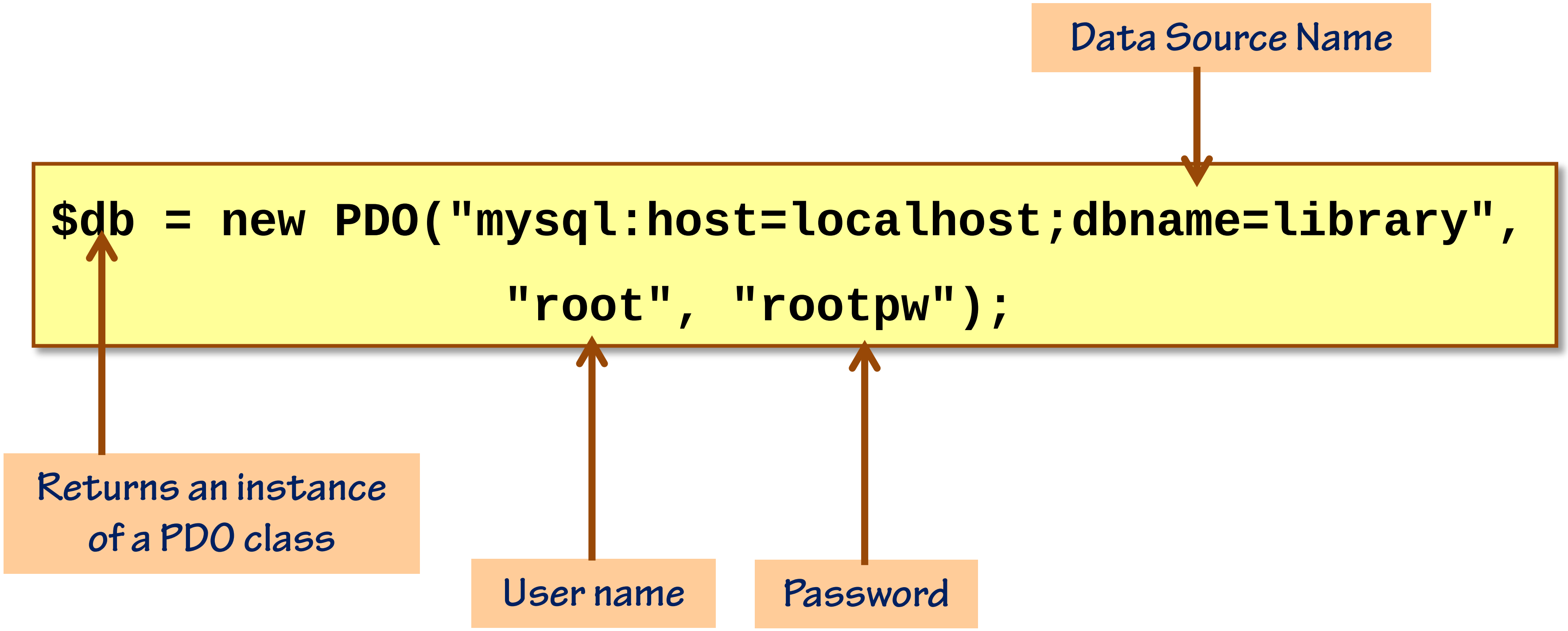
- **In this lesson we will use PDO (PHP Data Objects) to access a database:**
  - Log in to the database
  - Build and execute a query
  - Retrieve and display the results
  - Implement a book search page for our library
  - Validate user input
- **We will briefly compare PDO with `mysqli`**

Introducing PHP Data Objects (PDO)

- **PDO is a set of PHP extensions for accessing databases**
  - Object-Oriented
  - Supports error notification through exceptions
- **Requires a database-specific backend driver**
  - Drivers exist for most mainstream databases
- **Available from PHP 5.1 onwards**

Connecting to the Database

- The PDO class represents a connection to a database server



```
$db = new PDO("mysql:host=localhost;dbname=library",
                "root", "rootpw");
```

Methods of the PDO Class

- Some of the methods of the PDO class are shown below

Method	Description
`query`	Executes an SQL statement and returns a result set
`exec`	Executes an SQL statement and returns the number of affected rows
`quote`	Quotes a string for use in a query
`beginTransaction`	Initiates a transaction
`commit`	Commits a transaction
`rollback`	Rolls back a transaction
`errorInfo`	Gets extended error information associated with the last operation

Performing a Query

- Use the `query` method of the PDO class to run a "`select`" query

Run the query and return a result set

```
$sth = $db->query("select * from books");

while ($row = $sth->fetch(PDO::FETCH_ASSOC)) {
    printf("%s %s\n", $row["title"], $row["author"]);
}
```

Get the next row of the result set. Return `false` if there are no more rows

Return the result as an associative array

Index into the row using the column name

An Alternative way of Looping

- The result set returned by `$db->query` can be iterated using a `foreach` loop:

```
foreach($db->query($query) as $row) {
  printf("%s %s\n", $row["title"], $row["author"]);
}
```

PDO Error Reporting

- PDO provides three error-handling strategies

Strategy	Description
`PDO::ERRMODE_SILENT`	PDO just sets the error code; the program is responsible for checking it
`PDO::ERRMODE_WARNING`	PDO will emit a warning message
`PDO::ERRMODE_EXCEPTION`	PDO will set the error code and then throw a PDOException object which can be caught

class constants

Catching Exceptions

- Here's how to set the error strategy, and catch exceptions:

```
try {
  $db = new PDO( ... );
  $dbh->setAttribute(PDO::ATTR_ERRMODE,
                     PDO::ERRMODE_EXCEPTION);
  ...
}
catch (PDOException $e) {
  printf("We had a problem: %s\n", $e->getMessage());
}
```

Bringing it all Together

- Testing may be easier using stand-alone PHP code:

```php
<?php
try {
  $db = new PDO("mysql:host=localhost;dbname=library",
                "root", "rootpw");
  $sth = $db->query("select * from books " .
                    "where author like '%Bryson'");
  while ($row = $sth->fetch(PDO::FETCH_ASSOC)) {
    printf("%-40s %-20s\n", $row["title"], $row["author"]);
  }
}
catch (PDOException $e) {
  printf("We had a problem: %s\n", $e->getMessage());
}
?>
```

Query Results

- The script may be run explicitly from a terminal

```
$ php testit.php
Notes From a Small Island                  Bill Bryson
A Short History of Nearly Everything       Bill Bryson
A Walk in the Woods                        Bill Bryson
The Lost Continent                         Bill Bryson
```

Moving to a Web Application

- Re-casting the application as a web application involves extra code

Design a form to capture user input

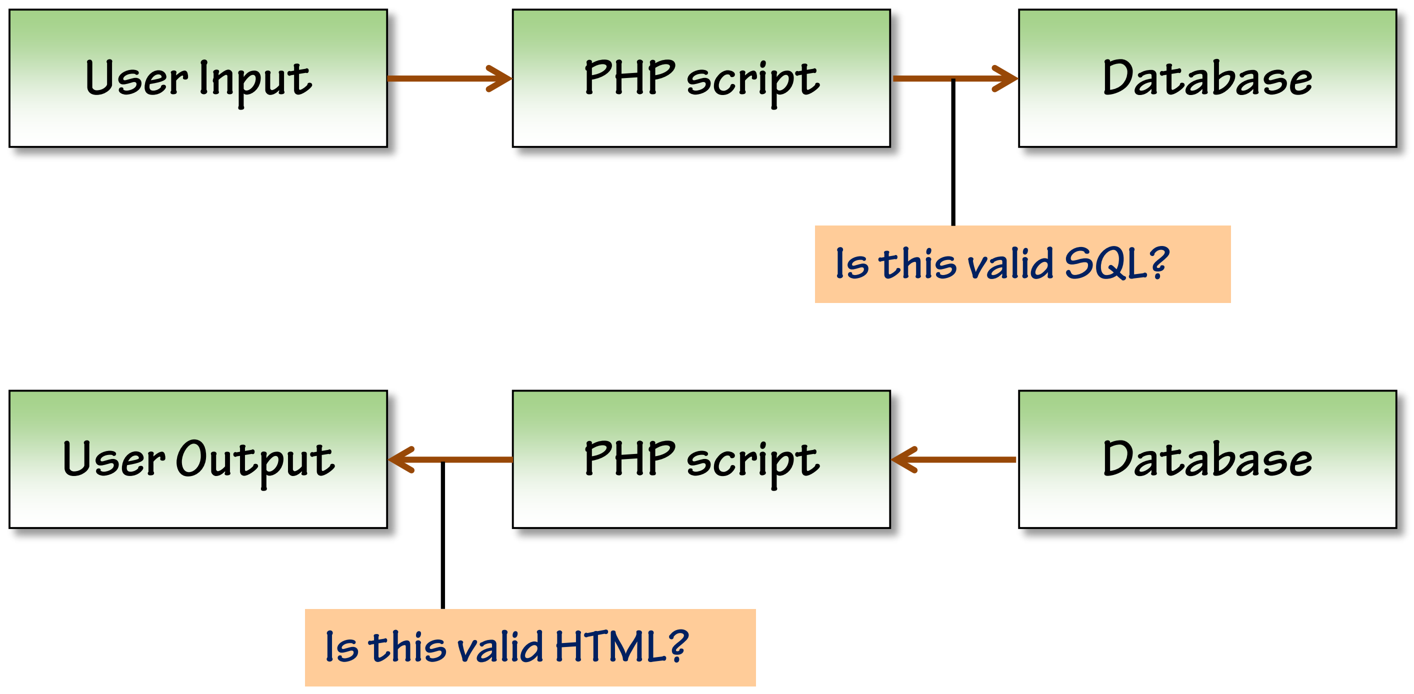
Add the outer HTML framework

Get user input from the form

Build the query

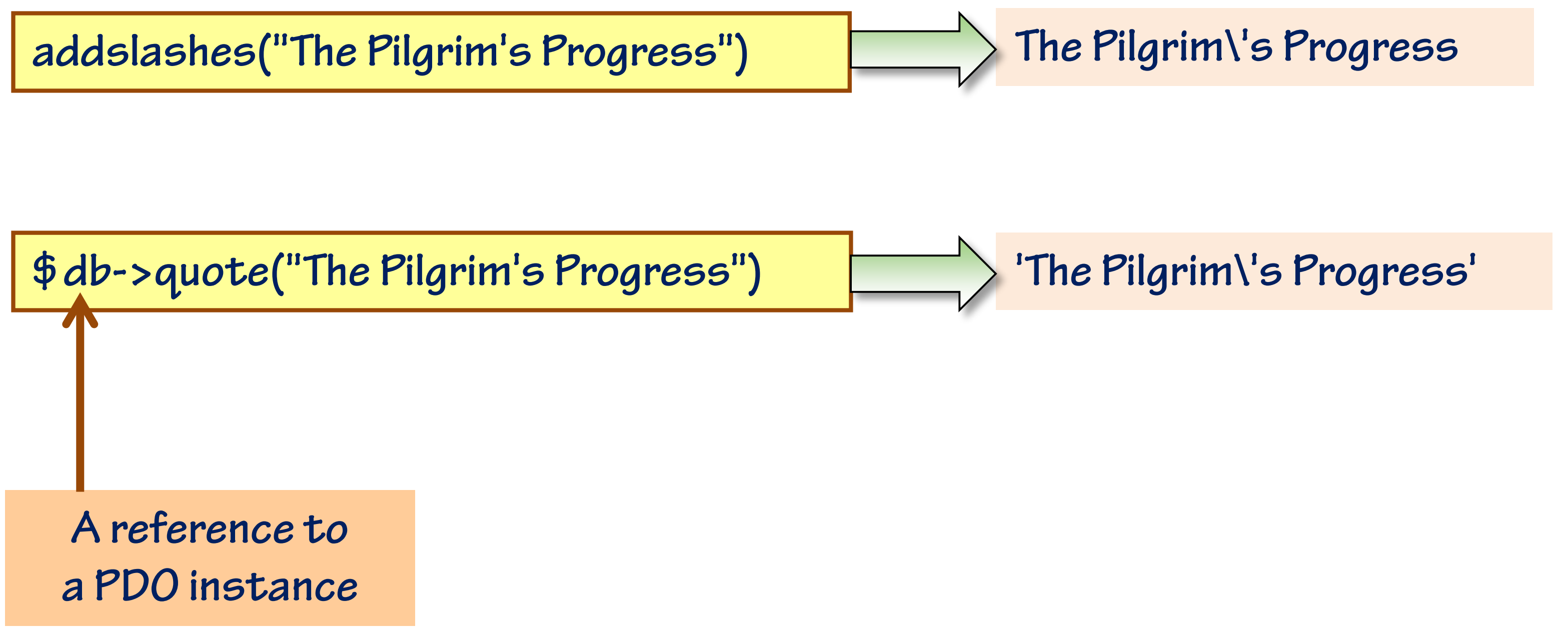
Return the results as an HTML table

Too many languages!



Quoting SQL

- **If text that we want to store in the database contains special characters, they need to be quoted**
  - Example: Book titles

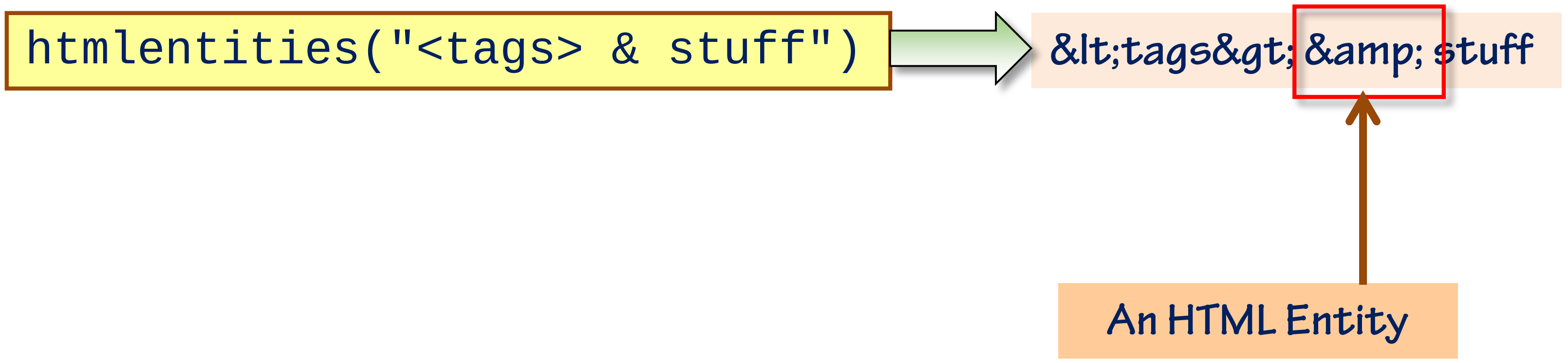


```
addslashes("The Pilgrim's Progress")  →  The Pilgrim\'s Progress

$db->quote("The Pilgrim's Progress")  →  'The Pilgrim\'s Progress'
```

HTML Entities

- **If text that we want to send to the browser contains special characters, they need to be converted to HTML Entities**
  - Example: Our library has a book by Andy Nerd called " <tags> & stuff "



```
htmlentities("<tags> & stuff")
```

```
&lt;tags&gt; &amp; stuff
```

Input Validation

- **It's important to validate user-supplied input**
  - To flag mistakes back to the user
  - To guard against malicious input
- **PHP's "filters" allow validation of a string against pre-defined types:**

```
$email = "fred@example.com";
if (! filter_var($email, FILTER_VALIDATE_EMAIL))
   echo "address invalid";
```

Filter types include:

```
FILTER_VALIDATE_BOOLEAN
FILTER_VALIDATE_EMAIL
FILTER_VALIDATE_FLOAT
FILTER_VALIDATE_INT
FILTER_VALIDATE_IP
FILTER_VALIDATE_URL
```

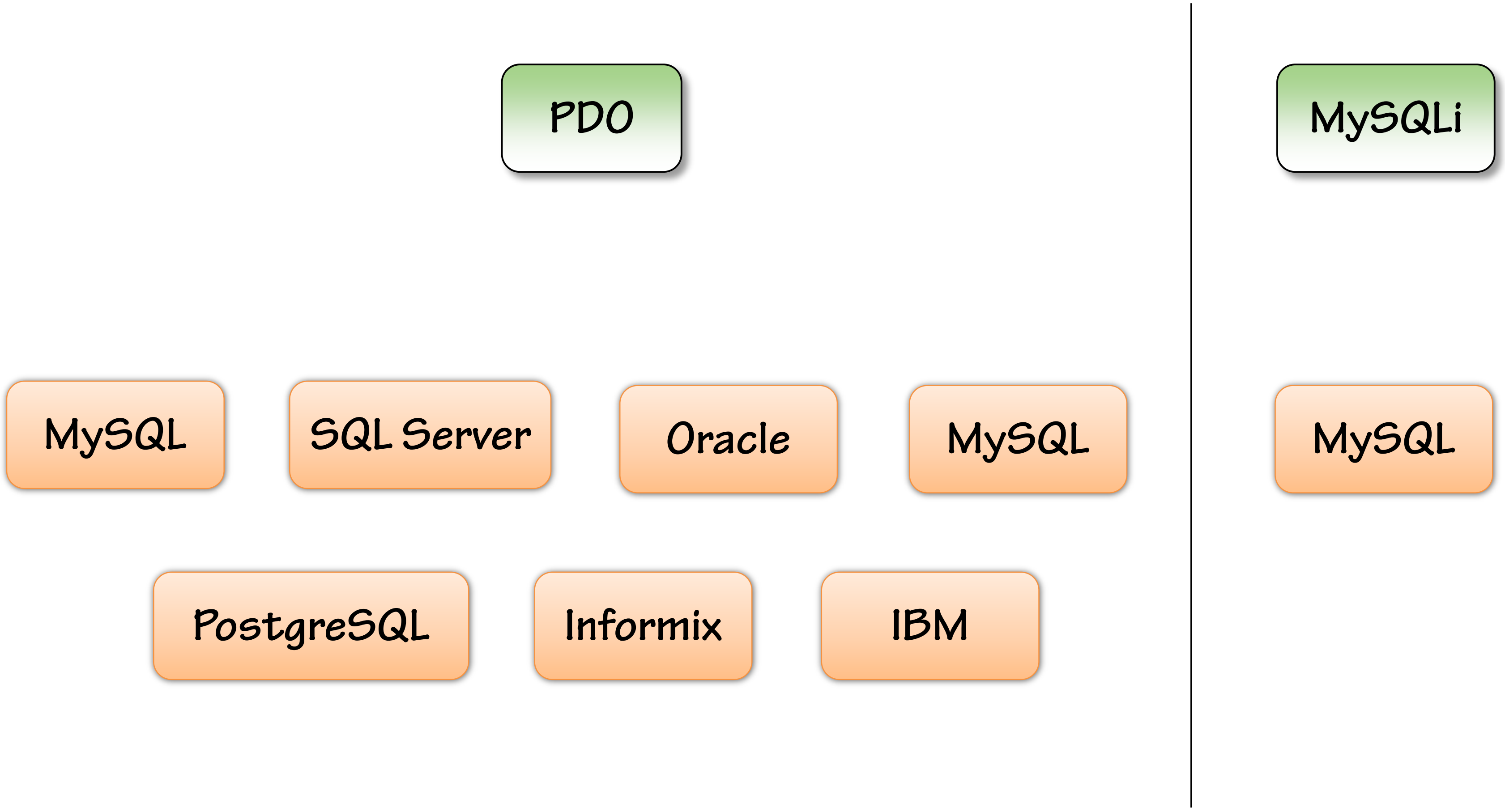
Introducing mysqli

- **The mysqli library provides an alternative way to access a database**
  - mysqli = "mysql improved"
- **The "improvements" include:**
  - An object-oriented interface
  - Support for prepared statements, multiple statements, and transactions
  - Better debugging capabilities
- **The original `mysql` library has been deprecated**

There is a comparison of mysql, myslqi and PDO at:

`http://php.net/manual/en/mysqlinfo.api.choosing.php`

Comparing `mysqli` with PDO



MySQLi: Procedural vs O-O Styles

- The `mysqli` library offers two styles of interface
- Procedural style:
  - Might be preferable if you're not used to an O-O programming style

```
$db = mysqli_connect("host", "user", "password", "database");
$res = mysqli_query($db, "select * from borrowers");
```

- Object-oriented style:
  - More like PDO

```
$db = new mysqli("host", "user", "password", "database");
$res = $db->query("select * from borrowers");
```

- Opinions differ about whether MySQLi or PDO is better!
  - We will choose PDO for the remainder of this course

Lesson Summary

- **We have used some key classes and methods of PDO**
  - Connect to the database
  - Execute a query
  - Retrieve the results
- **We must be careful to escape special characters so that they aren't mis-interpreted by the database or the browser**
- **`mysqli` provides an alternative to PDO**

Coming up in Lesson 6:

Doing more with the Database

Executing non-queries
Using prepared statements
Calling stored procedures