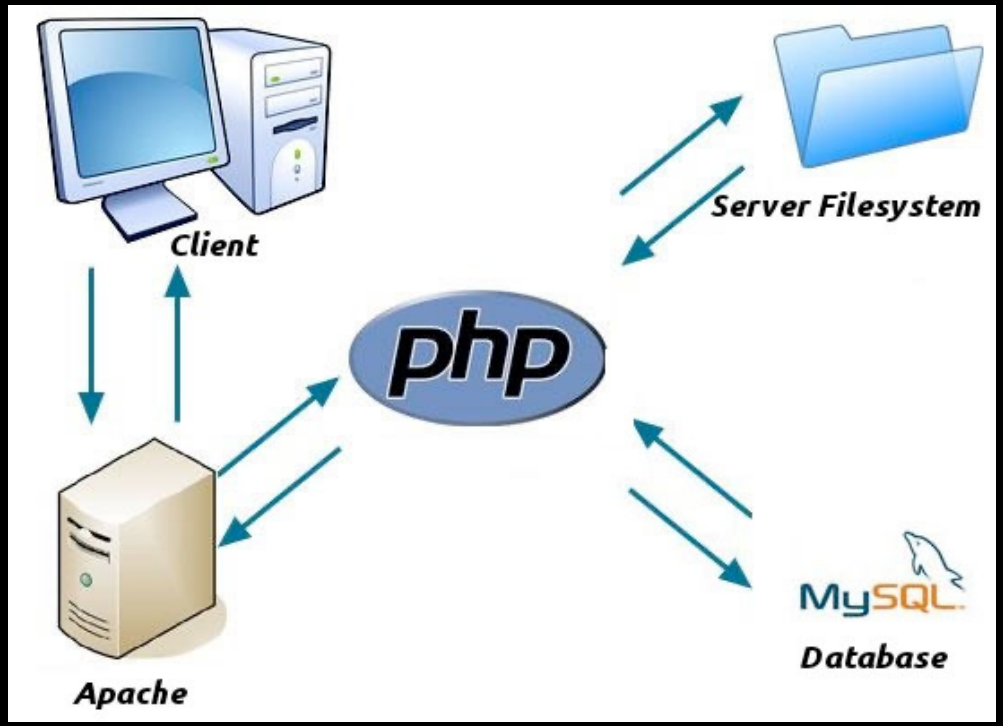
PHP and MySQL

Intro PHP

Download sample code from Nexus

PHP and MySQL on the Server

PHP and MySQL, along with the Apache web server, are at the heart of the LAMP stack, and have been so for a very long time.



Connecting to MySQL from PHP

In order to use MySQL, PHP must establish a connection to the MySQL server which runs as a daemon on the system, or sometimes even on a remote system.

The mysql connector

mysql_connect() and its related functions were deprecated in PHP 5, and were removed in PHP 7. If you see them in code, you know you are working on a legacy codebase.

The mysqli connector

mysqli and its related functions are available in both procedural and object oriented versions. It serves as a good transition to the modern fully object oriented PDO connector.

The PDO connector

PDO, or the PHP Data Objects, is a fully object oriented interface that provides consistent connections and methods to a variety of database including MySQL, Oracle, MS SQL, SQLite and more.

PDO Drivers

Table of Contents

The mysqli connector works only with MySQL. PDO, on the other hand, works with a large selection of modern databases.

For this reason, we will be using PDO from the start, even before we have covered Object Oriented PHP.

The Connection

Connecting to MySQL with PDO

To connect PHP to our MySQL server, we need to know a number of things. At a minimum...

The hostname where the MySQL server can be found. Most commonly localhost, but it could be on another domain entirely.

The name of the specific database we want to access

The username of the user who has privileges to access the database

The password for the connecting user

01_connect.php

Before we begin...

We need to setup a database, add some data, and create a user who can access it.

Open the setup.txt file, and copy the lines between start script and end script and paste them into MySQL. This will create our database and seed it with some sample data.

Now, we need to create a user with a password, and give that user access to our database. To do so, execute the following commands inside the MySQL shell:

```
create user 'web_user'@'localhost' identified by 'mypass';

grant all on borland.* to 'web_user'@'localhost';
```

01_connect.php

Create the connection in PHP

To connect to PHP, we need to configure the connection, and use that connection every time we want to communicate with our database.

```php
define('DB_DSN', 'mysql:host=localhost;dbname=borland');
define('DB_USER', 'web_user');
define('DB_PASS', 'mypass');

$dbh = new PDO(DB_DSN, DB_USER, DB_PASS);

$dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$dbh->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
```

```
object(PDO)[1]
```

01_connect.php

First Query

Start with the output

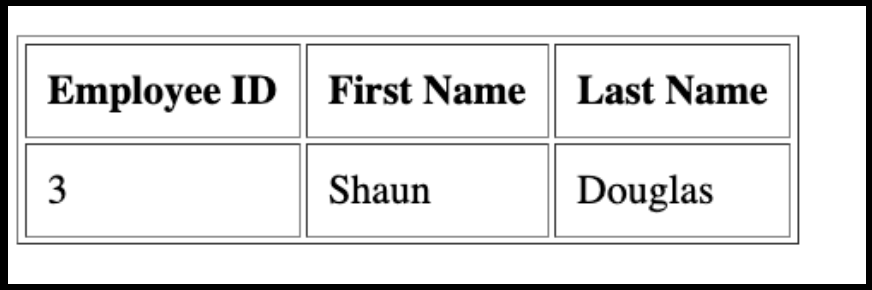
The easiest way to begin with PHP and MySQL, is to create the HTML for the output we want to see.

```
<table cellpadding="8" border="1">

<tr>
   <th>Employee ID</th>
   <th>First Name</th>
   <th>Last Name</th>
</tr>

<tr>
   <td>3</td>
   <td>Shaun</td>
   <td>Douglas</td>
</tr>

</table>
```



Employee ID	First Name	Last Name
3	Shaun	Douglas

02_hardcode_output.php

Create our first query in PHP

For our first query, we'll simply select all the fields we want to see in our output: id, first_name and last_name.

```
$query = "SELECT id, first_name, last_name FROM employees";

$stmt = $dbh->query($query);

$result = $stmt->fetchAll();
```

When we can potentially generate more than a single result, we use the fetchAll() method of the PDO Statement.

The next step is to modify our hard-coded output in the HTML table, to instead loop over the $results and output the actual field data.

Create our first query in PHP

Now that we've generated our results, we can convert our hard-coded table to display dynamic data.

```
<tr>
    <td>3</td>
    <td>Shaun</td>
    <td>Douglas</td>
</tr>
```

Remember, $results is a mutli-dimensional array. Each element is a row from the database. Each row, is an array containing our fields and data

The keys of each of the $row arrays match our database fields... we can use them to retrieve the data we want to output.

```
<?php foreach($result as $row) : ?>

<tr>
  <td><?=$row['id']?></td>
  <td><?=$row['first_name']?></td>
  <td><?=$row['last_name']?></td>
</tr>

<?php endforeach; ?>
```

03_first_query.php

Create our first query in PHP

For our first query, we'll simply select all the fields we want to see in our output: id, first_name and last_name.

```
<?php foreach($result as $row) : ?>

<tr>
  <td><?=$row['id']?></td>
  <td><?=$row['first_name']?></td>
  <td><?=$row['last_name']?></td>
</tr>

<?php endforeach; ?>
```

```
<tr>
    <td>1</td>
    <td>Dave</td>
    <td>Jones</td>
</tr>


<tr>
    <td>2</td>
    <td>Henry</td>
    <td>Bissoon</td>
</tr>


<tr>
    <td>3</td>
    <td>Margaret</td>
    <td>Thomson</td>
</tr>


<tr>
    <td>4</td>
    <td>Jill</td>
    <td>King</td>
</tr>
```

ID	First Name	Last Name
1	Dave	Jones
2	Henry	Bissoon
3	Margaret	Thomson
4	Jill	King

03_first_query.php

Refactor the code

Since we'll be using our PDO connection a lot, let's refactor the script and put it in an include.

```
require __DIR__ . '/inc/functions.php';
require __DIR__ . '/inc/connect.php';

$query = "SELECT id, first_name,
          last_name FROM employees";

$stmt = $dbh->query($query);

$result = $stmt->fetchAll();
```

With the PDO connection information now in an external file, we can include it in every PHP script where we need to make a database connection.

While we're at it, if we include our functions file, we can utilize our e() function to properly escape our data for output.

```
<?php foreach($result as $row) : ?>

<tr>
  <td><?=e($row['id'])?></td>
  <td><?=e($row['first_name'])?></td>
  <td><?=e($row['last_name'])?></td>
</tr>

<?php endforeach; ?>
```

04_refactor_code.php

Prepared Statements

Prepared statements

Prepared statements allow us to escape data before use in an SQL query. This is the third context of FIEO in PHP.

```php
// The user we want to see
$id = 2;

$query = "SELECT *
          FROM employees
          WHERE
          id = ?";

$stmt = $dbh->prepare($query);

$stmt->bindValue(1, $id, PDO::PARAM_INT);

$stmt->execute();

$emp = $stmt->fetch();
```

Our first query was easy... we simply selected all records. But if we need to filter the results, or access a single record, we may need to allow the user to pass in some parameters to the PHP script.

In this example, we've hard coded the parameter $id... but in our next script, we'll make it dynamic, and allow the user to pass in the $id.

05_prepared_statement.php

Prepared statements

Prepared statements allow us to escape data before use in an SQL query. This is the third context of FIEO in PHP.

```
// The user we want to see
$id = 2;

$query = "SELECT *
          FROM employees
          WHERE
          id = ?";
```

The ? mark is a placeholder that will be replaced by the value of $id when the query is executed.

```
$stmt = $dbh->prepare($query);
```

The next step is to allow MySQL to prepare the query for execution. It will know to expect a value to replace the ?.

```
$stmt->bindValue(1, $id, PDO::PARAM_INT);
```

Now we bind the $id to the first placeholder (1), and cast it as an integer (PDO::PARAM_INT).

```
$stmt->execute();
```

The next step is to execute the query, with its bound data.

```
$emp = $stmt->fetch();
```

In this case, because we're only expecting a single result, we can use the fetch() method to retrieve it.

05_prepared_statement.php

Output the single result

With a single result retrieved using fetch(), the result is a one dimensional array – no loop is necessary.

```
<p>
    <strong>Employee ID</strong>: <?=e($emp['id'])?><br />
    <strong>First Name</strong>: <?=e($emp['first_name'])?><br />
    <strong>Last Name</strong>: <?=e($emp['last_name'])?><br />
    <strong>Email</strong>: <?=e($emp['email'])?><br />
    <strong>Phone</strong>: <?=e($emp['phone'])?><br />
    <strong>Department</strong>: <?=e($emp['department'])?>

</p>
```

05_prepared_statement.php

Make the query dynamic

Showing the employee with the hard-coded $id of 2 is not very useful. We need to allow the user to choose the $id.

```
// The user we want to see
if(empty($_GET['id'])) die('Please enter an employee id');

// Make sure the user provided an integer
if(intval($_GET['id']) != $_GET['id']) die('Employee id must be an integer');
```

http://example.com/employees?id=1
http://example.com/employees?id=3
http://example.com/employees?id=5
http://example.com/employees?id=22

Simply by retrieving the $id from the query string of a GET request ($_GET['id']), we can allow the user to determine which single record is retrieved and displayed.

06_dynamic_record.php

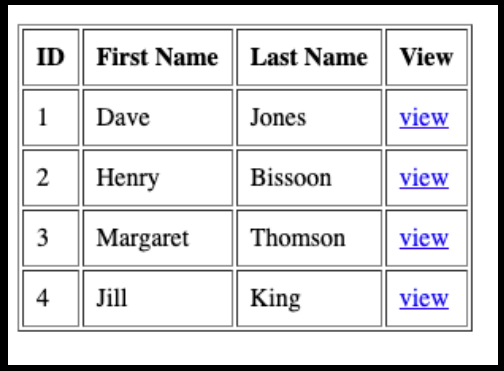
Connecting list and detail views

We can't ask users to manually enter a user id in the GET query string. To make things easy, we must connect our list view and detail view.

```
<?php foreach($result as $row) : ?>

<tr>
  <td><?=e($row['id'])?></td>
  <td><?=e($row['first_name'])?></td>
  <td><?=e($row['last_name'])?></td>
  <td><a href="08_detail_view.php?id=<?=e($row['id'])?>">view</a></td>
</tr>
```

If we add a link to our output of the list of employees, we can simply add the ?id= at the end of it. Now, all the user has to do is click the link to view a record.



ID	First Name	Last Name	View
1	Dave	Jones	view
2	Henry	Bissoon	view
3	Margaret	Thomson	view
4	Jill	King	view

07_list_view.php

Creating a simple search function

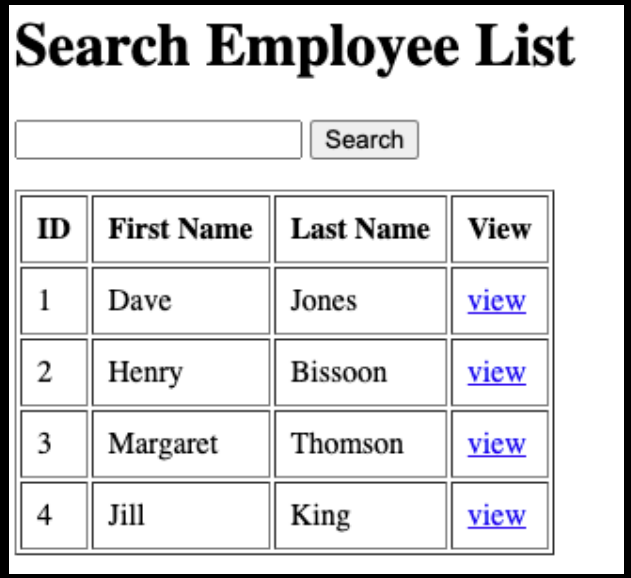
We can make our small application even more dynamic by adding search functionality to our list view.

```
<form>
    <input type="search" name="search" size="20” value="<?=e($search)?>" />
    <input type="submit" value="Search" />
</form>
```

```
<?php if(!$result) : ?>
 <h3>Your search returned no results</h2>
<?php else : ?>
```

If there are no results from the search, we output a message, else we will show the list view.

08_search.php



Creating a simple search function

We can make our small application even more dynamic by adding search functionality to our list view.

```php
$search = $_GET['search'] ?? '';

if(!empty($search)) {

      $query = "SELECT id, first_name, last_name
                  FROM employees
                  WHERE
                  first_name LIKE :search
                  OR
                  last_name LIKE :search";

      $stmt = $dbh->prepare($query);

      $stmt->bindValue(':search', "%$search%", PDO::PARAM_STR);

} else {
      $query = "SELECT
                  id, first_name, last_name
                  FROM employees";

      $stmt = $dbh->prepare($query);
}
```

We conditionally change our query depending on whether or not the user has provided a search term in the GET query string.

Note the use of named placeholders instead of the positional ? placeholders.

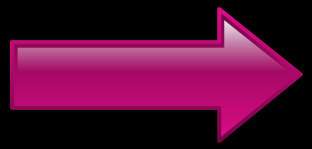
This allows us to re-use the placeholder for the same parameter, and to bind it only once.

08_search.php

The Building blocks of the web

Connected list and detail views are the building blocks of the web. Master them, and you can build anything.

ID	First Name	Last Name	View
1	Dave	Jones	view
2	Henry	Bissoon	view
3	Margaret	Thomson	view
4	Jill	King	view



Single Record

Employee ID: 1
First Name: Dave
Last Name: Jones
Email: djones@borland.com
Phone: 212-555-3456
Department: Sales

08_search.php

Next:
PHP and MySQL Inserts