Lab # 06: Interfacing PHP with MySQL

OBJECTIVES OF THE LAB

This lab aims at the understanding of:

- *Type of MySQL Drivers in PHP*
- *PHP-MySQL Interfaces*

TYPES OF MYSQL DRIVERS IN PHP

This section introduces the options available to you when developing a PHP application that needs to interact with a MySQL database.

Application Programming Interface (API)

An Application Programming Interface, or API, defines the classes, methods, functions and variables that your application will need to call in order to carry out its desired task. In the case of PHP applications that need to communicate with databases the necessary APIs are usually exposed via PHP extensions.

APIs can be procedural or object-oriented. With a procedural API you call functions to carry out tasks; with the object-oriented API you instantiate classes and then call methods on the resulting objects. Of the two the latter is usually the preferred interface, as it is more modern and leads to better organized code.

When writing PHP applications that need to connect to the MySQL server there are several API options available. This lab discusses what is available and how to select the best solution for your application.

Connector

Connector refers to a piece of software that allows your application to connect to the MySQL database server. MySQL provides connectors for a variety of languages, including PHP.

If your PHP application needs to communicate with a database server you will need to write PHP code to perform such activities as **connecting to the database server**, **querying the database** and **other database-related functions**. Software is required to provide the API that your PHP application will use, and also handle the communication between your application and the database server, possibly using

other intermediate libraries where necessary. This software is known generically as a connector, as it allows your application to connect to a database server.

Driver

A driver is a piece of software designed to communicate with a specific type of database server. The driver may also call a library, such as the MySQL Client Library or the MySQL Native Driver. These libraries implement the low-level protocol used to communicate with the MySQL database server.

By way of an example, the PHP Data Objects (PDO) database abstraction layer may use one of several database-specific drivers. One of the drivers it has available is the PDO MYSQL driver, which allows it to interface with the MySQL server.

Extension

The PHP code consists of a core, with optional extensions to the core functionality. PHP's MySQL-related extensions, such as the mysqli extension, and the mysql extension, are implemented using the PHP extension framework.

An extension typically exposes an API to the PHP programmer, to allow its facilities to be used programmatically. However, some extensions which use the PHP extension framework do not expose an API to the PHP programmer.

The PDO MySQL driver extension, for example, does not expose an API to the PHP programmer, but provides an interface to the PDO layer above it.

What are the main PHP API offerings for using MySQL?

There are three main API options when considering connecting to a MySQL database server:

- PHP's MySQL Extension
- PHP's MYSQLi Extension
- PHP Data Objects (PDO)

PHP MySQL Extension

This is the original extension designed to allow you to develop PHP applications that interact with a MySQL database. The mysql extension provides a procedural interface and is intended for use only with

MySQL versions older than 4.1.3. This extension can be used with versions of MySQL 4.1.3 or newer, but not all of the latest MySQL server features will be available.

Note:
If you are using MySQL versions 4.1.3 or later it is strongly recommended that you use the mysqli extension instead.

PHP MySQLI Extension

The mysqli extension, or as it is sometimes known, the MySQL improved extension, was developed to take advantage of new features found in MySQL systems versions 4.1.3 and newer. The mysqli extension is included with PHP versions 5 and later.

The mysqli extension has a number of benefits, the key enhancements over the mysql extension being:

- Object-oriented interface
- Support for Prepared Statements
- Support for Multiple Statements
- Support for Transactions
- Enhanced debugging capabilities
- Embedded server support

PHP Data Objects (PDO)

PHP Data Objects, or PDO, is a database abstraction layer specifically for PHP applications. PDO provides a consistent API for your PHP application regardless of the type of database server your application will connect to. In theory, if you are using the PDO API, you could switch the database server you used, from say Firebird to MySQL, and only need to make minor changes to your PHP code.
0
While PDO has its advantages, such as a clean, simple, portable API, its main disadvantage is that it doesn't allow you to use all of the advanced features that are available in the latest versions of MySQL server. For example, PDO does not allow you to use MySQL's support for Multiple Statements.

Comparison of Features

Table 6.1 compares the functionality of the three main methods of connecting to MySQL from PHP. It can be seen that either PHP's MYSQLI or PDO interface must be used for new projects. MySQLI comes in both procedural and object-oriented interfaces. It is recommended to use MYSQLI object-oriented interface.

PHP-MYSQL INTERFACTING

This section provides interfacing of PHP with its all three extensions.

TABLE 6.1 COMPARISON OF MYSQL API OPTIONS FOR PHP

	PHP's mysqli Extension	PDO (Using PDO MySQL Driver and MySQL Native Driver)	PHP's MySQL Extension
PHP version introduced	5.0	5.0	Prior to 3.0
Included with PHP 5.x	yes	yes	Yes
MySQL development status	Active development	Active development as of PHP 5.3	Maintenance only
Recommended by MySQL for new projects	Yes - preferred option	Yes	No
API supports Charsets	Yes	Yes	No
API supports server-side Prepared Statements	Yes	Yes	No
API supports client-side Prepared Statements	No	Yes	No
API supports Stored Procedures	Yes	Yes	No
API supports Multiple Statements	Yes	Most	No
Supports all MySQL 4.1+ functionality	Yes	Most	No

Example 1: Using PHP-MYSQL Interface

```
<?php
    //step 1. Database Connection
    //To connect to database: server name, user, and user's password
    //must be provided.
    $db=mysql_connect("localhost","root","");

    //step 2. Database Selection
    //To select database: provide database name and connection variable
    $db_select=mysql_select_db("pvfc_db",$db);
?>

<html>
<head>
    <title> PHP MySql Basics </title>
</head>
```

```
16. <body>
17.     <?php
18.         //step 3. Perform query or operation on database
19.         $result=mysql_query("SELECT * FROM department", $db);
20.
21.         //step 4. Display query results
22.         while($row=mysql_fetch_array($result)){
23.             echo $row[0]." ".$row[1]<br />";
24.         }
25.     ?>
26. </body>
27. </html>
28.
29. <?php
30.     //step 5. Close Database Connection
31.     mysql_close($db);
32. ?>
```

This example demonstrates the interfacing of PHP using MYSQL procedural interface. There are five main steps. In first step, connection is established with MySQL. In second step, database is selected. In third step, various operations on selected database tables are performed. In fourth step, query results are displayed and in the fifth step, database connection is closed. This interfacing is shown for completion purpose. It is recommended to use MYSQLI Object-Oriented Interface.

Example 2: Using PHP-MYSQLI Procedural Interface

```
1.  <?php
2.      //step 1. Database Connection + Database Selection
3.      //server name, user, user's password, and database
4.      //must be provided.
5.      $db = mysqli_connect("localhost", "root", "", "department");
6.
7.      //step 2. Check Connection
8.      if(mysqli_connect_errno())
9.      {
10.         echo "Failed to connect to MySql: ".mysqli_connect_error();
11.     }
12. ?>
13.
14. <!DOCTYPE html>
15. <html>
```

```
<head>
    <title>PHP MySqli (Procedural) Basics</title>
</head>
<body>
    <?php
            //step 3. Perform query or operation on database
            $sql = "SELECT * FROM department";
            $result = mysqli_query($db, $sql);

            //step 4. Show result
            while($row = mysqli_fetch_array($result, MYSQLI_ASSOC)){
                printf ("%s %s\n",$row["dep_ID"],$row["dep_Name"]);
     ?>
    <br/>
    <?php }
    //step 5. Free result set
    mysqli_free_result($result);
?>
</body>
</html>

<?php
    //step 6. Close Database Connection
    mysqli_close($db);
?>
```

This example demonstrates the interfacing of PHP using MYSQLI procedural interface. There are six main steps. In first step, connection is established with MySQL and database is selected. In second step, database connection is checked for errors. In third step, various operations on selected database tables are performed. In fourth step, query results are displayed. In fifth step, result variable is freed. In the sixth step, database connection is closed. This interfacing is again shown for completion purpose. It is recommended to use MYSQLI Object-Oriented Interface.

Example 3: Using PHP-MYSQLI Object-Oriented Interface

```
<?php
    //step 1. Database Connection + Database Selection
    //server name, user, user's password, and database
    //must be provided.
    $servername = "localhost";
    $username = "root";
    $password = "";
```

```
8.      $database = "pvfc_db";
9.
10.     // Create Connection
11.     $conn = new mysqli($servername, $username, $password, $database);
12.
41.     //step 2. Check Connection
13.     if ($conn->connect_error) {
14.         die("Connection failed: " . $conn->connect_error);
15.     }
16.     echo "Connected successfully";
17. ?>
18.
19. <!DOCTYPE html>
20. <html>
21. <head>
22.     <title>PHP MySqli (Object-Oriented) Basics</title>
23. </head>
24. <body>
25.     <?php
26.             // query code
27.             //step 3. Perform query or operation on database
28.             //step 4. Show results
29.             //step 5. Free result set
30.     ?>
31. </body>
32. </html>
33.
34. <?php
35.     //step 6. Close Database Connection
36.     $conn->close();
37. ?>
```

This example demonstrates the interfacing of PHP using MYSQLI object-oriented interface. There are six main steps. In first step, connection is established with MySQL and database is selected. In second step, database connection is checked for errors. In third step, various operations on selected database tables are performed. In fourth step, query results are displayed. In fifth step, result variable is freed. In the sixth step, database connection is closed.

Example 4: Using PHP-PDO Interface with

```
1. <?php
2.      //step 1. Database Connection + Database Selection + Check Connection
```

```
3.      //server name, user, user's password, and database
4.      //must be provided.
5.      $username = 'root';
6.      $password = '';
7.      try
8.      {
9.          $str = 'mysql:host=localhost;dbname=pvfc_db;charset=utf8';
10.         $db = new PDO($str,$username, $password);
11.         $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
12.         echo "Connected successfully";
13.         echo "</br>";
14.         echo "</br>";
15.     }
16.     catch(PDOException $e)
17.     {
18.         echo "Connection failed: " .$e->getMessage();
19.     }
20. ?>
21.
22. <!DOCTYPE html>
23. <html>
24. <head>
25.     <title>Php-PDO Basics </title>
26. </head>
27. <body>
28.     <?php
29.         //step 2. Perform query or operation on database
30.         $q = $db->prepare("select * from department");
31.         $q->execute();
32.
33.         //step 3. Show result
34.         $q_records = $q->fetchAll();
35.         foreach ($q_records as $row){
36.             printf ("%s %s\n",$row["dep_ID"],$row["dep_Name"]);
37.             echo "</br>";
38.         }
39.     ?>
40. </body>
41. </html>
42.
43. <?php
44.     //step 4. Close Database
45.     $db = null;
```

46. ?>

This example demonstrates the interfacing of PHP using PDO. There are four main steps. In first step, connection is established with MySQL; database is selected; and database connected is checked for errors. In second step, various operations on selected database tables are performed. In third step, query results are displayed. In fourth step, database connection is closed.

----------Task 6.1----------

Complete Example 3 by providing a suitable query, its result, and then freeing result variable.

----------Task 6.2----------

Write PHP-MYSQLI Object-Oriented interface script that connects with one of the tables of your DBMS Lab Project (For instance, user table or accounts table or any of your choice). Perform the following:

a. Apply the insert command on the table. Take the data from user via HTML Form constructs.
b. Apply the update command on the table. Take the data from user via HTML Form constructs.
c. Apply the delete query on the table. Take the data from user via HTML Form constructs.
d. Apply the select query on the same table to retrieve data. Show the data in HTML Form Constructs.

----------Task 6.3----------

Perform Task 6.2 using PHP-PDO (PHP Data Objects) interface.

----------Task 6.4----------

Perform Task 6.2 using LARAVEL.