Lecturer: Zainab Khyioon Abd alrdha

The msqli extension

The mysqli extension features a dual interface. It supports the procedural and object-oriented programming paradigm.

Users migrating from the old mysql extension may prefer the procedural interface. The procedural interface is similar to that of the old mysql extension. In many cases, the function names differ only by prefix. Some mysqli functions take a connection handle as their first argument, whereas matching functions in the old mysql interface take it as an optional last argument.

Mixing styles

It is possible to switch between styles at any time. Mixing both styles is not recommended for code clarity and coding style reasons.

What is an 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 terms API and extension should not be taken to mean the same thing, as an extension may not necessarily expose an API to the programmer.

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)**

Each has its own advantages and disadvantages.

What is PHP's 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.

The *mysql* extension source code is located in the PHP extension directory `ext/mysql`.

What is PHP's 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**

Note:
If you are using MySQL versions 4.1.3 or later it is *strongly* recommended that you use this extension.

As well as the object-oriented interface the extension also provides a procedural interface.

The *mysqli* extension is built using the PHP extension framework, its source code is located in the directory `ext/mysqli`

What is 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.

Other examples of database abstraction layers include JDBC for Java applications and DBI for Perl.

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.

PDO is implemented using the PHP extension framework, its source code is located in the directory `ext/pdo`.

Comparison of Features

The following table compares the functionality of the three main methods of connecting to MySQL from PHP:

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	Yes -	Yes	No

Comparison of MySQL API options for PHP			
	PHP's mysqli Extension	PDO (Using PDO MySQL Driver and MySQL Native Driver)	PHP's MySQL Extension
MySQL for new projects	preferred option		
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

Using MySQLi with PHP with an OO approach

Introduction

MySQLi is the new interface incorporated within PHP language, where I stands for improved. The improved version of the interface can be used with versions from MySQL 4.1 on and up. It is possible to follow both the object oriented and the procedural approach.

Connect to DB

To connect to a MySQL server, the following syntax is to be used:

```php
<?php

$host = 'localhost';
$user = 'yourusername';
$pass = 'yourpassword';
$dbname = 'databasename';

// OOP way
 $db = new mysqli($host, $user, $pass, $dbname);
// Procedural way would be:
 $db = mysqli_connect($host, $user, $pass, $dbname);

?>
```

It is possible to use an error suppressor (the @ symbol) before your instructions. This error suppressor should only be used when you are able to catch an error on your own, and display an error message to the user. This suppressor has a warning in PHP.net:

Reference:
http://www.php.net/manual/en/language.operators.errorcontrol.php

Currently the "@" error-control operator prefix will even disable error reporting for critical errors that will terminate script execution. Among other things, this means that if you use "@" to suppress errors from a certain function and either it isn't available or has been mistyped, the script will die right there with no indication as to why.

Carrying on with our topic, as you can see, we have instantiated the mysqli class. Since this instantiation returns an object, we can now invoke the methods of this class. In turn, utilizing the procedural way, this returns a resource, thus representing the database connection. This means that each time you will call a mysqli function, this same resource is needed, which will indicate, what connection you are referring to.

Displaying an error

Since we have suppressed the error in this case, we have a function that tells us if the connection was successful or not. This function can be invoked the same way

for both procedural and object oriented way. In my opinion, it 'looks' like a standalone function. To display an error to the user, we do it like this:

```php
<?php
if(mysqli_connect_errno())
{
    die('The connection to the database could not be
established.');
}
?>
```

And that's how simple it is to display an error to the user, and exit the process, since our logic will not run correctly due to the missing database connection.

Changing databases

If at a certain point, you need to change the database, you can do so with the following:

```php
<?php
// OOP way
$db->select_db($new_dbname);
// Or the procedural way
// We send the paramater $db as the resourse
mysqli_select_db($db, $new_dbname);
?>
```

Running a query

Let's create a query to retrieve all the users and their information to display it on the screen. To do so, we would setup a string with our query, and send it to the function like this:

```php
<?php
// Set up query
$query = 'SELECT * FROM users';
// OOP way
$result = $db->query($query);
// Procedural way
$result = mysqli_query($db, $query);
```

```
?>
```

Remember that when using the OOP way, your returned result will be an object, as to the procedural way, you will get another resource. If the above query is not successful for any reason, our result will be a simple 'FALSE'.

Get the results

Obviously, if we have just queried a database, it means that we need this data. If you would like to know how many records this query has returned, then we can go ahead and use the attribute num_rows. We achieve this with a simple line of code:

```
<?php
// OOP way
$total_results = $result->num_rows;
// Procedural way
$total_results = mysqli_num_rows($result);
?>
```

Now that we know how many results are, we can go ahead and loop through the results, and display them on the screen. Let's go ahead and use a while loop. The fetch_assoc function will return an array for each record that was found, and each will have each key as their attributes, and each value in the array.

```
<?php

echo 'There are $total_results record(s) found';
// OOP way
while ($row = $result->fetch_assoc())
{
    echo '<p>';
    echo $row['username'].' ';
    echo $row['firstname'].' ';
    echo $row['lastname'].' ';
    echo $row['city'].' ';
    echo $row['state'].'</p>';
}

// Procedural way
```

```
while ($row = mysqli_fetch_assoc($result))
{
    echo '<p>';
    echo $row['username'].' ';
    echo $row['firstname'].' ';
    echo $row['lastname'].' ';
    echo $row['city'].' ';
    echo $row['state'].'</p>';
}
?>
```

If you wanted to grab the results as an object, you could just use the fetch_object function.

```
<?php
// OOP way
$row = $result->fecth_object();
// Procedural way
$row = mysqli_fetch_object($result);
// Attributes are accessed in the following:
$row->username;
$row->firstname; // Etc...
?>
```

Close connection

Although PHP automatically closes your connection upon script termination, if you want to close the connection before your script is done, you can do so by just invoking the close function. This is done by doing the following. First though, you should 'free' up the result identifier, which will free up the memory. Then use the close function, to close the connection.

```
<?php
// OOP way
$result->free();
$db->close();
// Procedural way
mysqli_free_result($result);
mysqli_close($db);
```

```
?>
```