Using MySQL with PDO

Peter Lavin

revised July 25, 2011 (originally published on the MySQL site)

Table of Contents

Prerequisites for Using PDO

PDO is a PHP extension providing a data-access abstraction layer that can be used with a variety of databases. This gives you the flexibility of changing the database backend without having to alter your access methods. Even if you use MySQL exclusively, PDO can provide advantages; you can use the same data-access methods regardless of the MySQL version. This does away with the need for using the standard MySQL extension with older versions of MySQL and using the MySQLi extension for later versions. An additional advantage is the ability to use object-oriented code regardless of the version of MySQL.

PDO requires the object-oriented capabilities of PHP 5, so PHP 5.0 or higher is a prerequisite. The PDO extension ships with binary versions of PHP 5.1 and 5.2 and is very simple to implement on most operating systems.

Compiling PHP from source is the one sure way to customize PHP to your exact specifications and ensure that you have not only PDO but also the drivers you need. However, the package managers of most current Linux distributions make it easy to add support—if it's not already there.

Note

Some earlier versions of Mac OS X do not include the PDO driver.

The PDO extension is entirely object-oriented—there is no procedural version of this extension so some knowledge of object-oriented programming is assumed.

Project Outline

This article examines a project that uses PDO to select from and insert into a MySQL database of PHP classes. The PDO classes are as follows:

- `PDO` – the PDO connection object

- `PDOStatement` – the PDO statement object returned by the connection `query` method or the `prepare` method
- `PDOException` – the PDO-specific exception object
- `PDORow` – a representation of a result set row as an object

All PDO classes are used with the exception of the `PDORow` class.

XML representations of PHP classes, both internal and user-defined, are transformed into SQL INSERT statements and added to a MySQL database. This is done using the SimpleXMLElement class, one of the built-in PHP classes. Even if your grasp of XML is elementary, you'll find the SimpleXMLElement easy to understand and use.

The attached compressed file contains all the code and preserves the required directory structure. Find that file here. Any user-defined classes are found in the `classes` directory and XML files are found in the `xml` directory. The PHP scripts are found at the root directory.

The XML Document Format

Using PHP's reflection classes you can automate the process of converting a PHP class to XML. We won't concern ourselves here with the details of how to do this; we'll review an example file in this section and include other examples in the `xml` directory. The reflection classes allow you to introspect your own classes or built-in classes. They are useful tools for reverse engineering code and discovering the properties of objects at runtime. More importantly perhaps, they provide a way of generating documentation from source code.

If you haven't used the reflection classes before you can quickly get a sense of their capabilities by executing `Reflection::export( new ReflectionClass( ClassName));`. This static method of the Reflection class dumps all the methods, data members, and constants of the ReflectionClass object passed to it.

The following example is an XML representation of what reflection reveals about the `mysqli_sql_exception` class.

```
<?xml version="1.0"?>
<classobj>
  <name>mysqli_sql_exception</name>
  <documenting_date>2007-07-31</documenting_date>
  <php_version>5.2.0</php_version>
  <type final="" abstract="">class</type>
  <origin mod_date="" num_lines="">internal</origin>
  <parent_class>RuntimeException</parent_class>
  <class_doc_comments/>
  <interfaces_list/>
  <methods_list>
    <method static="0" final="1" abstract="0" declaring_class="Exception" priority="2">
      <method_name>__clone</method_name>
      <method_origin>internal</method_origin>
      <visibility>private</visibility>
      <method_doc_comment/>
    </method>
    <method static="0" final="0" abstract="0" declaring_class="Exception" priority="2">
      <method_name>__construct</method_name>
      <method_origin>internal</method_origin>
      <visibility>public</visibility>
      <param classtype="" defaultvalue="" byreference="" isoptional="1">message</param>
      <param classtype="" defaultvalue="" byreference="" isoptional="1">code</param>
      <method_doc_comment/>
      ...
    </method>
  </methods_list>
  <datamembers_list>
    <datamember visibility="protected" static="0" defaultvalue="""">
      <datamember_name>message</datamember_name>
      <datamember_doc_comment/>
    </datamember>
    ...
  </datamembers_list>
  <constants_list/>
```

```
</classobj>
```

Not all methods or data members of the `mysqli_sql_exception` class are included but there's enough detail here to form an idea of what any XML representation of a PHP class might look like.

Most of the tags and attributes are self explanatory. For example, by looking at the `message` data member you can readily determine that it is a protected, non-static data member with no default value. Details will become more apparent if you examine the DDL scripts used to create the database tables. Find these statements in the attached compressed file.

Designing the Tables

An XML class file is effectively a flat-file database. To convert this file to a relational database format requires a number of tables. These are:

- Classes (including interfaces)
- Parent interfaces
- Methods
- Method parameters
- Data members
- Constants

To view the SQL needed to create these tables find the `phpclasses.sql` script in the `scripts` directory. You may want to look at this file as each of the tables is discussed in the following sections.

The Classes Table

The most obvious table required is the table of classes and interfaces. Find below the SQL necessary to create this table.

The `name` field uniquely identifies each record so it could be used as a primary key but an AUTO_INCREMENT field is more convenient. Only two modifiers can be applied to a class or interface and these are `final` and `abstract` . (Static classes don't exist in PHP so there's no need for this modifier and all classes are public so there is no need for a visibility specifier.)

The `type` and `origin` fields are candidates for the ENUM data type since, in both cases, there are only two options; object templates are either classes or interfaces and these are either user-defined or built-in. The `last_modified` ,`num_lines` and `class_doc_comment` fields only apply to user-defined classes since reflection cannot capture this information when used with an internal class.

The Parent Interfaces Table

PHP doesn't support multiple inheritance for classes so a simple `parent_class` field is all that's required in the classes table. On the other hand, multiple interfaces can be implemented; hence the need for an interfaces table.

This is a simple table made up of the id number of the implementing class and the interface name.

The Methods Table

Classes can also have any number of methods, requiring a table similar to the classes table.

Like the table of classes, this table supports a `method_origin` field and the modifiers, final and abstract. However, methods can also be static and have a visibility modifier. The `declaring_class` field simply indicates whether a method is inherited as is, overridden in the child, or entirely new.

Unlike some other object-oriented languages, method overloading (in the sense of two methods with the same name but a different number or different types of parameters) is not supported in PHP. For this reason, a primary key may be created from the class id and method name.

The Parameters Table

Methods may have any number of parameters, hence the need for a parameters table.

Parameters may have default values, may be optional, and may be passed by value or by reference. As of PHP 5.1 parameters may also be type-hinted so a `class_type` field is also necessary. Parameter variable names must be unique to the method and method names must be unique to a class so these two fields along with the class id uniquely identify a parameter and form the primary key.

The Data Members and Constants Tables

The data members table incorporates a number of fields common to the other tables and is a simpler version of the methods table.

A table of constants is much simpler than the data members table and is made up of three fields only, the `class_id` , `constant_name` , and its associated value.

Unlike data members, any comments that accompany constants cannot be retrieved so there is no `doc_comment` field for the table of constants.

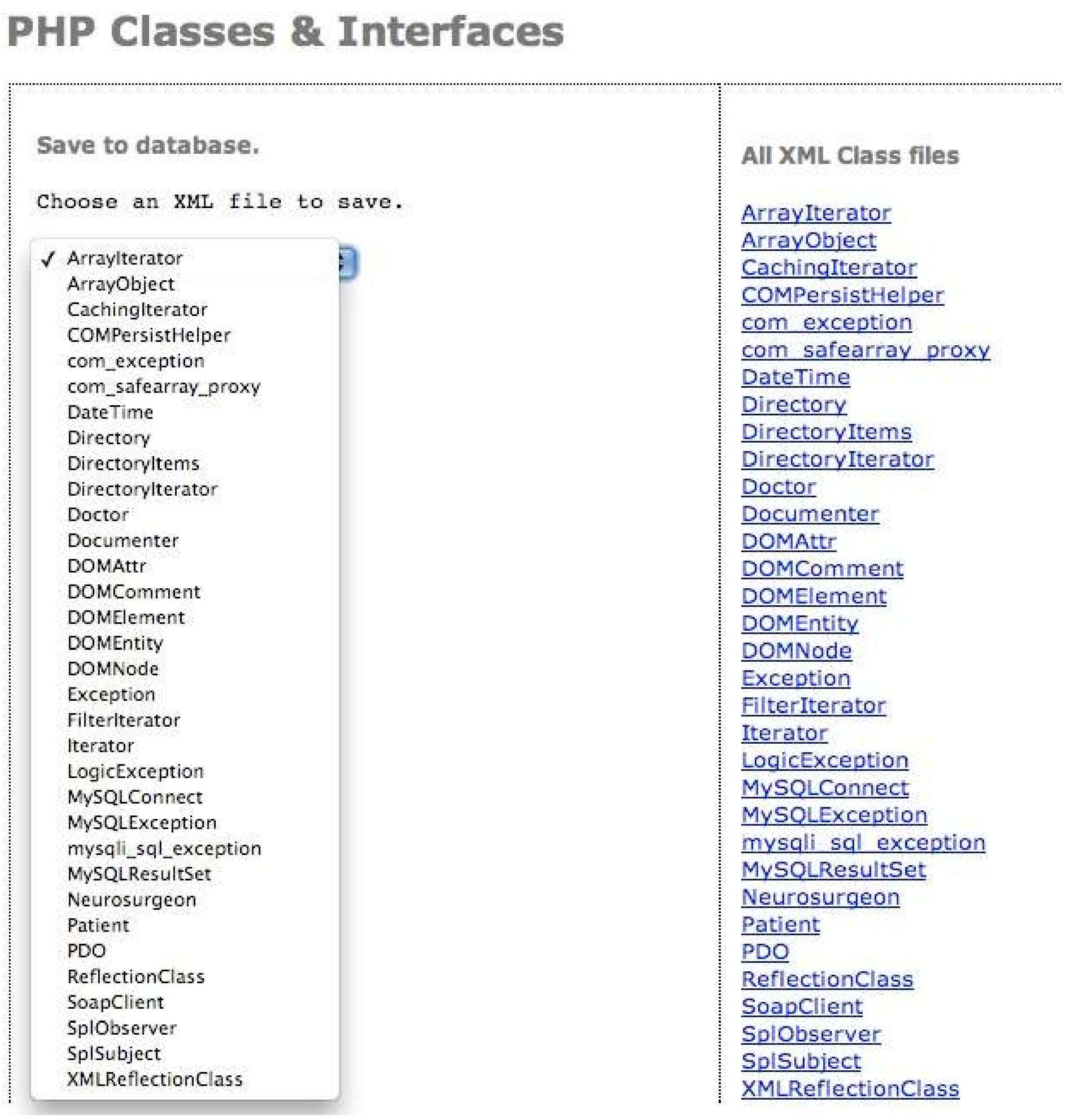
Creating the Tables

The script file `phpclasses.sql` contains the DDL statements for creating the database, `phpclasses`, and the tables. Execute this script from the command line by navigating to the `scripts` directory and executing `mysql -u` *`user_name`* `-p`*`password`* `< phpclasses.sql`.

Creating a PDO Connection

Records are added to a database using a web form. Within this form a drop-down list box is populated with the names of classes that are not already included in the database, thereby eliminating duplicate submissions. The result is pictured below:

Figure 1. Select control



The elements of the list box match the XML files on the right because no XML files have been added to the database yet.

Existing XML class files are found in a directory named `xml`. The XML file name (excluding the `xml` extension) matches the corresponding PHP class name. These names are copied from the `xml` directory into an array using a user-defined class, `DirectoryItems` . This array is compared to the classes already contained in the database to create a drop-down list of classes (the `<select>` control in the following HTML) that aren't yet included. The code to do this follows.

```
<form name= "select_class" action="add_record.php" method="get" style="padding:5px;">
  <select style="max-width: 180px; min-width: 180px;" name = "xmlfilename" >
    <?php
      //autoload user-defined classes
```