Starting to use ODBC with PHP

A beginner's guide to universal database connectivity using PHP

Daniel J Lewis

May 24, 2011

Learn how to set up and use the universal database connection technique known as Open Database Connectivity in conjunction with the web development language PHP. This is an introductory article for the technologies required to make an open, reliable, and standardized web application — namely, PHP, ODBC, and SQL.

PHP is one of the most commonly used programming languages for dynamic website development. It is reasonably powerful and efficient. It also happens to be quite simple and can be a great language for beginners because of its flexibility.

PHP is good as a language on its own (especially when coupled with XHTML). However, the majority of applications need a convenient data-storage method, typically fulfilled by a database, such as MySQL or PostgreSQL. For PHP to connect to data-storage systems, a connector is required that allows functions like `mysql_query()` and `pg_query()`. This works under most circumstances, but occasionally, business requirements demand that data be stored in something like a spreadsheet that can be more easily manipulated by individuals. In these situations, a different type of connector is required to talk to the data.

What if you want to send SQL to no particular database system and want your SQL to work with any data store you configure? This is the very reason why Open Database Connectivity (ODBC) was created and, with the right connectors installed, you can even access Microsoft® Excel®, CSV, and other data file types. ODBC is a connector that makes PHP development "database connector-agnostic." It uses functions like `odbc_query()` against databases like MySQL, PostgreSQL, SQLite, Microsoft SQL Server®, IBM® DB2®, Sybase, OpenLink Virtuoso, FileMaker, and Microsoft Office® Access®. It can also be used against CSVs and Excel spreadsheets, depending on the correct ODBC driver setup.

**ODBC: What is it?**

ODBC is a software driver system for connecting programming languages with data stores. It is a free and open source system that appeared in 1992 and attempts to standardize connection methods, such as functions and configuration, across programming languages and database query access (SQL standardization).
ODBC works by acting as a twofold interface or connector design: First, as a programming language system to an ODBC system and second, as an ODBC system to a data storage system. Therefore, ODBC requires a programming-language-to-ODBC driver (for example, the PHP-ODBC Library) and an ODBC-to-data-storage-system driver (such as the MySQL-ODBC Library). This is in addition to the ODBC system itself, which handles the configuration of data sources and allows the ambiguity of data sources and programming languages.

**How do PHP and SQL fit in?**

PHP is a programming language, generally used as a server-side language for powering dynamic websites. It is flexible by being a dynamic and weakly typed language. It is oddly familiar to many developers because it was influenced by the C programming language. A free and open source programming language that appeared in 1995, you can use PHP with databases via a connector to generate XHTML and HTML for rendering in a web browser.

SQL is a cross-platform language for interrogating a data store. It is primarily used for relational databases, but it does have procedural, object-oriented, and object-relational extensions. Modern implementations of SQL can be found within MySQL, PostgreSQL, SQLite, DB2 (both commercial and Express-C editions), Microsoft SQL Server, OpenLink Virtuoso, FileMaker, and Microsoft Access, all of which you can connect to using programming languages like PHP through a connectivity system (ODBC).

**Setting up ODBC**

Let's look at how a typical Linux-Apache-PHP-MySQL (LAMP) environment can be transformed into this flexible Linux-Apache-PHP-ODBC (LAPO) environment. There are two general options for ODBC drivers on Linux®: iODBC and unixODBC. Both sets of drivers have their own pros and cons, and both work with PHP, but they work with different sets of databases. My choice is iODBC because of its traditionally strong connection to web programming languages like PHP and Ruby, as well as its stability in working with ODBC-friendly databases like MySQL and OpenLink Virtuoso. However, it is just a matter of choice, and you may want to look into your efficiency requirements. Other than subtle internal differences, both iODBC and unixODBC work in the same way in terms of connecting with a programming language — the PHP functions are identical — and the database (for example, MySQL would not be affected).

Both iODBC and unixODBC are available in Linux Software Package Managers. For instance, on the Debian, Ubuntu, or Linux Mint command line, you would run `sudo apt-get install iodbc`.

**ODBC in other environments**

Apple Mac OS X ships with iODBC preinstalled, so there’s no need to install anything. Windows® operating systems come with an ODBC driver of their own, as well.

**Connecting the database with ODBC**

If you haven't already, install a database system, such as MySQL or PostgreSQL. Then install the ODBC-to-database connector. This connector will vary from database to database, but taking MySQL as an example, install the connector by installing the operating system-relevant driver from the MySQL website.
In Linux distributions that support `apt`, you can run the following command from the console: `sudo apt-get install libmyodbc`.

You will have to configure your ODBC client to work with your installed database by running a program such as iodbcadm-gtk. You can also edit your iODBC file manually. (In Linux, this file is often in `/etc/iodbc.ini`.)

**Example: connecting PHP with ODBC**

Next, you must install the PHP ODBC driver. You can do so by adding iODBC or unixODBC into the PHP compilation script (very complex) or by installing the PHP-ODBC Library. In an apt-based distribution, you can use the following command: `sudo apt-get install php5-odbc`.

Test the flow by running PHP in interactive mode (`php -a`). This opens the PHP interactive console, where you can interact in a way similar to the example in Listing 1.

**Listing 1. Command-line ODBC connection**

```php
php > $conn = odbc_connect(
    "DRIVER={MySQL ODBC 3.51 Driver};Server=localhost;Database=phpodbcdb",
    "username", "password");
php > $sql = "SELECT 1 as test";
php > $rs = odbc_exec($conn,$sql);
php > odbc_fetch_row($rs);
php > echo "\nTest\n--\n" . odbc_result($rs,"test") . "\n";
Test ---- 1
php > odbc_close($conn);
php > exit;
```

Let's analyze the code from Listing 1:

1. Establish a connection using the `odbc_connect()` function in PHP. This function takes an ODBC connector string, a user name, and a password as parameters. The connector string will be matched with your odbc.ini file, so make sure that it matches beforehand.
2. Instantiate a variable to string, which represents your SQL statement.
3. Execute your SQL using `odbc_exec`, which takes in your connection and your SQL string and returns a result set.
4. Fetch just one row from that result set using `odbc_fetch_row()`, which takes in the result set as a parameter. This is an iterative function, meaning that if you call it again, you'll get the next result in the set (and so on, until there are no results in the result set, in which case, it will return false).
5. Use the function `odbc_result()`, which takes in a result set and a column name (as a string) and returns the cell value within the row that the row iterator points to.
6. Close the ODBC connection using `odbc_close`, which takes in the connection itself.
7. Quit PHP interactive mode by sending an `exit` command.

This is great, but it isn't much use on a web application scale. For you to test this in a client/server-style web browsing mode, you need to have a web server, such as Apache or Lighttpd, installed.
(If you’re running Linux, make sure you get the PHP module for your web service; otherwise, PHP won’t run.)

Listing 2 shows the same techniques as executed through a web server. The PHP code is similar to what we did in Listing 1, but exports the results through XHTML rather than on the command line.

**Listing 2. XHTML-based ODBC connection example**

```php
<?php
$conn = odbc_connect(
    "DRIVER={MySQL ODBC 3.51 Driver};Server=localhost;Database=phpodbcdb",
    "username", "password"); 
if (!($conn)) {
    echo "<p>Connection to DB via ODBC failed: 
    " . odbc_errormsg ($conn) . "</p>
    " .
    echo "<p>Connection to DB via ODBC failed: " .
    echo odbc_errormsg ($conn) . "</p>
    " .
    odbc_close($conn); 
    echo "</p>\n";
} 
$sql = "SELECT 1 as test";
$rs = odbc_exec($conn,$sql);
if (!($conn)) {
    echo "<p>Connection to DB via ODBC failed: 
    " . odbc_errormsg ($conn) . "</p>
    " .
    echo "<p>Connection to DB via ODBC failed: " .
    echo odbc_errormsg ($conn) . "</p>
    " .
    odbc_close($conn); 
    echo "</p>\n";
} 
while (odbc_fetch_row($rs)) {
    $result = odbc_result($rs,"test"); 
    echo "<tr>$result</td><td></tr>"
} 
php
</html>
```

There is an addition to this listing that wasn’t in Listing 1: You’re now taking full advantage of odbc_fetch_row() being an iterative function by placing it within a `while` loop. This means that if your SQL were a bit more complicated and queried a table for multiple results, it would create a new line in the rendered XHTML table.

There are many types of XHTML and HTML, with various amounts of browser support and ease of use. Listing 2 generates standardized XHTML 1.0 Strict, which is one of the best forms of XHTML for developing data-centric and strong cross-browser documents.

**PHP-ODBC programming**

There are four main types of ODBC functions: functions for connection, querying, fetching data, and error reporting. The querying functions are capable of coping with the standard database transactions for creating, reading, updating, and deleting data (known as CRUD operations).
Connection functions

Every process that has started must have an end; otherwise, you get memory and processor issues. Therefore, make sure you close your database connections.

You've already seen that `odbc_connect()` takes in an ODBC-friendly connection string, a database user name, and the relevant password. It returns a connection object you can use throughout your PHP program. The following code shows an example:

```php
$connection = odbc_connect($connection_string, $username, $password);
```

You have also already seen in a previous example that `odbc_close()` takes in a connection object and terminates communications with ODBC and the database. I can't stress enough that you must close your connections; otherwise, you'll have too many connections to your database and have to restart your database management system — or even worse, your whole computer. This is how it works: `odbc_close($connection);`

Query functions

You've used `odbc_exec()` already; it takes in a connection object and an SQL string, fires it all off, and returns a result set object. A result set object is a complicated beast, as it usually resides within the memory of the database management system and can only be deciphered by functions that interact with it. An `odbc_exec()` line looks a bit like this: `$resultset = odbc_exec($connection, $sql);`

The `odbc_prepare()` and `odbc_execute()` functions are incredibly useful for injecting unknown variables into SQL. The `odbc_prepare()` function prepares an SQL statement with the database management system, then `odbc_execute()` sends in the variables. This means it's powerful, more secure, and more efficient than simply building up a concatenated SQL string in PHP and sending it through `odbc_exec()`. Together, they look a bit like this:

```php
$resultset = odbc_prepare($connection, $sql);
$succes = odbc_execute($resultset, $variables);
```

Listing 3 is a nice example I created to search for people in a users table, based on a location and date-of-birth variables. Note the question mark (?) in the SQL string, which denotes the variables defined in the serial array of the `odbc_execute()` function.

Listing 3. SQL variable injection with prepare and execute commands

```php
$location = "London";
$mindateofbirth = time() - 567648000; /* i.e. 18 years ago from now */
$resultset = odbc_prepare(
    $connection,
    "SELECT * FROM user WHERE location = ? AND dateofbirth <= ?"
);
$succes = odbc_execute($resultset, array($location, $mindateofbirth));
```
Fetching functions

The `odbc_fetch_row()` takes in a result set from a query and shifts an iterator pointer from one row to the next. This function is often used in conjunction with the `odbc_result()` function to fetch various cells:

```php
odbc_fetch_row($resultset);
```

In earlier examples, the `odbc_result()` takes in a `$resultset` and a column name string and returns the value of a cell. This is used in conjunction with the `odbc_fetch_row()` function to point at a particular row in the result set:

```php
$value = odbc_result($resultset, "columnname");
```

The `odbc_fetch_array()` function is somewhat similar in that it is an iterator function used to fetch data from the result set of a query. However, this time, it returns an array that represents the row with the column name as the key and the cell as the value:

```php
$rowarray = odbc_fetch_array($resultset);
```

Similar to `odbc_fetch_array()`, the `odbc_fetch_object()` instead fetches an object-oriented structure representing the row. It places column names as object properties and cell values as the property values:

```php
$rowobject = odbc_fetch_object($resultset);
```

This function is useful for printing a set of results in HTML. It is a simple rendering of the results but can be useful when prototyping or debugging:

```php
odbc_result_all($resultset);
```

The `odbc_num_fields()` function is a rather nice function that simply takes in a result set and tells you the number of rows in it:

```php
$numberofrows = odbc_num_rows($resultset);
```

Problem solving and debugging

Two handy PHP ODBC functions — `odbc_error()`, which returns an error code if an error has occurred or false if no error has occurred, and `odbc_errormsg()`, which returns a user-friendly message — can be combined to form a simple error messaging sequence:

```php
if (odbc_error()) {
    echo "I've found a problem: " . odbc_errormsg($conn);
}
```

One additional tip if you are getting errors while developing is never to be afraid of adding print statements near lines that are causing problems — provided you delete these "debug lines" when showing others, of course. Be aware of the following PHP function, which is often a lifesaver:
print_r($variable);

This simple function takes in any variable and displays it to the screen. The variable could be as simple as an integer or a string or as complex as a multidimensional array or an object.

**Universal connectivity**

Consider building a web application that could be deployed anywhere (examples of such could include Drupal, WordPress, or Joomla). These are often built using one database (for example, MySQL) and its database-specific functions (for instance, `mysql_connect()`), then carefully re-crafting it to work with another database (for instance, PostgreSQL) by changing functions (for instance, `pg_connect()`). This is a redundant practice when using ODBC, as the configuration is purely in initialization of the application, and the ODBC functions are androgynous to the database system.

One thing to be aware of, however, is that although all database management systems share a standardized SQL, they sometimes also include their own extensions. This is why it is a bit trickier to convert an existing PHP-MySQL, PHP-PostgreSQL, or PHP-MS-SQL application into a PHP-ODBC application. So when building an application from scratch, you must be careful to use strictly standardized SQL (or at most very common extensions of SQL).

As mentioned, it is possible to use ODBC to connect to a spreadsheet. As with the databases, you must do this using a connector. There are many out there — some are open source, but many are proprietary. One example is Microsoft Office for Windows, which comes with ODBC connectors for Excel spreadsheets. It can be quite awkward to deal with spreadsheets via ODBC, and it will probably solve a lot of hassle if you convert a simple spreadsheet into a database table. However, once a spreadsheet ODBC connection is established, you can treat it more or less like a database connection — the same ODBC PHP functions, but with a SQL-like language and, in Excel, using standard spreadsheet formula.

**Implications to linked data and the semantic web**

The linked data movement looks at connecting data across the web. Doing so has many benefits — primarily for machines understanding particular elements of data, but also for users, such as enabling them to find information more easily. The linked data movement uses pre-existing standards from the semantic web (such as the Resource Description Framework and the Web Ontology Language) and Internet/web standardization (such as HTTP and OpenID). It is beginning to be understood that the linked data connection methods are somewhat similar to ODBC in that a URI is somewhat similar to a connection string, and the Semantic Web Query Language (SPARQL) being somewhat similar to SQL through the established connection.

Extending the theory that linked data is somewhat analogous to ODBC, it would be possible to establish an ODBC connection to a linked data store (such as a "triple store") and send SPARQL queries down an ODBC connection. This has been the case with OpenLink Virtuoso, which allows you to connect via the standard ODBC connection.
Licensing and ODBC

iODBC is dual-licensed under the GNU General Public License (GPL) and Berkeley Software Development open source licenses. UnixODBC, too, is licensed under the GPL open source license. This means that whatever you develop with these two libraries does not have to be open source, but can be proprietary. Microsoft's ODBC driver can also be part of proprietary software, but is bound by the licensing agreements of Microsoft Office software (in the case of the Access databases and Excel spreadsheets) and SQL Server (in the case of SQL Server databases).

Conclusion

ODBC can be a great technique for maximizing universal connectivity. It enhances efficiency and allows you to expand applications to work with new forms of data, such as the web-based linked data. It does come with its downsides, however: In order to achieve universal connectivity, you must carefully choose the way you build your SQL queries because only a subset of all available SQL commands can be used across all database management systems. Hopefully, this article has provided what you need for starting to work with databases through ODBC using the PHP programming language.
Related topics

- The **W3schools section on SQL** is useful as a beginner's tutorial and as an advanced reference point for SQL. It details what is standardized as well as what works on one database management system and what works on another.
- The **W3Schools section on PHP** is another useful resource for developers starting with PHP as well as those PHP developers who need a reference point for the language.
- **iODBC.org** provides a lot of information on the installation and setup of iODBC, but also contains some useful information on generic ODBC and the connectors to various database systems and programming languages.
- **unixODBC.org** provides information on the installation, setup, and connector drivers for unixODBC.
- **MySQL.org** is the home of the MySQL Database Management System for developers, which has been part of the Oracle family since January 2010.
- You can get the **MySQL ODBC Connector** direct from MySQL.
- **Microsoft ODBC driver details on MSDN** and in the help files of the ODBC Data Source Administrator program. You'll find in these two locations details of working with Access, SQL Server, and Excel through ODBC.
- **PostgreSQL.org** is the home of the PostgreSQL database management system.
- A **PostgreSQL ODBC Connector project** is available as free and open source software.
- **OpenLink Virtuoso** is a "universal server." It provides web hosting (including with the PHP language), a database, ODBC connectivity, and semantic web technology. OpenLink Software are also the chief maintainers of the open source iODBC library, which comes in open source and proprietary versions.
- For web servers other than OpenLink Virtuoso, check out **Apache HTTP Server** and **Lighttpd Server**.
- **PHP.net** is the home for everything PHP.
- You will find details about all the PHP **ODBC functions** in the PHP.net documentation.
- **LinkedData.org** is a good portal for further information about linked data, linking open data, and the semantic web. The **W3C Semantic Web portal** can also be a useful resource.
- **SQL Standards** are held by the International Organization for Standardization.
- **developerWorks podcasts**: Listen to interesting interviews and discussions for software developers.