PDO_4D Driver

By Jesse Pina, Technical Services Team Member, 4D Inc.

Technical Note 10-12

Table of Contents

Abstract

The PDO_4D driver is a brand new product that enables Apache and IIS web servers to easily access 4D databases by way of PHP. 4D has sponsored an Open Source project, with the goal being the PDO_4D driver, which can be installed in any web server that uses PHP and can be used to run SQL code and 4D methods. The purpose of this Technical Note is to explain: the different pieces involved, how the pieces work together, and how to use the PDO_4D driver. Some examples are included and since PHP is being used, some basic PHP information is also included.

Introduction

PDO_4D has sponsored an Open Source project to create a PDO driver. The resulting driver is called PDO_4D and it enables 3rd party web servers such as Apache and IIS to access 4D databases by way of PHP. Creating a PDO driver has been made possible by the addition of the SQL Engine within 4D. The PDO_4D driver allows for PHP code to use SQL to access a 4D database's SQL Server.

Besides the fact that you can now more easily use 3rd party web servers to access 4D databases, the PDO_4D driver also allows for coding compartmentalization. With PDO_4D, you can now have a 4D expert focus on the database coding and have PHP expert focus on writing the frontend PHP code.

This Tech Note will give an overview of the different technologies involved, how they work together, and how to use the PDO_4D driver. Some examples are included and since PHP is being used, some basic PHP information will be included. Also, since this may be the first time that some developers are exposed to 3rd party web servers, an example is included that steps though the setup of: Apache, PHP, and PDO_4D.

What is PDO

Definition

First here are a few official definitions from http://PHP.net:

- PHP – a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.
- PDO – The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. Each database driver that implements the PDO interface can expose database-specific features as regular extension functions. Note that you cannot perform any database functions using the PDO extension by itself; you must use a database-specific PDO driver to access a database server.

PDO is a PHP extension that serves as a data-access abstraction layer. In other words, it removes the need for database system specific code and simplifies the PHP commands to basically writing a SQL statement. So whichever database system is ultimately used in the end (4D, mySQL, Oracle, SQLite, …), the interface is going to be the similar, if not the same. In this case, the term interface means the actual PHP code that is written to access the database. The PDO extension, basically allows for drivers to be built for specific databases, while keeping the interface (PHP code) the same.

So in order to access a specific database system, in this case 4D, a PDO driver will need to be created, and this is the goal of the PDO_4D open source project.

Uses

One of the main goals of PDO is to help developers write code that will access a variety of databases (4D, MySQL, SQLite, …) with little to no database specific code being written. The idea is to have the same PHP code run regardless of the database that is being accessed, with the exceptions being: the actual connection information and possible stored procedure calls. Also, while the PHP code may not vary, each database system may have a different SQL implementation, so the SQL code may be different as well.

A common problem that arises with developing web applications is the fact they require significant time and effort to abstract the database layer out as much as possible. Then on top of that, a large amount of database specific code still needs to be written for the specific database (4D, MySQL, SQLite, …) that you want to use with the web application. PDO basically provides the data-access abstraction layer, so the developer won't have to.

Who would use PDO

Before getting too far along, one important thing to note is that the PDO_4D driver is intended to be used more by PHP developers, rather than by 4D developers. The specific "database" code used amounts to SQL code. The requirements for writing code that uses the PDO_4D driver are essentially: 1) basic knowledge of PHP; 2) basic knowledge of SQL. You need to know PHP in order to build the web page and you need to know SQL to be able to access the database elements.

One of the advantages of PDO_4D is that you can now separate the web coding from the database coding. You can have 4D developers focusing on writing the backend code and you can have PHP developers focusing on writing the frontend code.

Project Details

The PDO_4D Open Source project currently only makes the source code available. The actual binaries are expected to be delivered by the PHP Group or the community from PECL Tools and available Distributors. The PHP driver that was used in the examples in this Tech Note was built using the version 0.2.1 of the source that was released on 09-01-2009. The source code package for this version and all other versions can be downloaded at the following location: http://pecl.php.net/package/PDO_4D.

Besides providing the source code, the packages will contain instructions for how to build a PDO_4D driver, as well as the appropriate license information.

PDO_4D: Requirements, Setup, and Installation

PDO_4D was created to allow PHP to access 4D databases. As stated above, PDO_4D implements the PDO interface to help developers write PHP code that accesses a 4D database.

Requirements

In order to use a PDO_4D driver, the following 3 items are required:

- A database running using 4D or 4D Server with version 12. The SQL Server must be running.
- A web server that supports PHP, such as Apache or IIS
- PHP version 5.2.0 or higher with modules mbstring and PDO 1.0.0 or newer

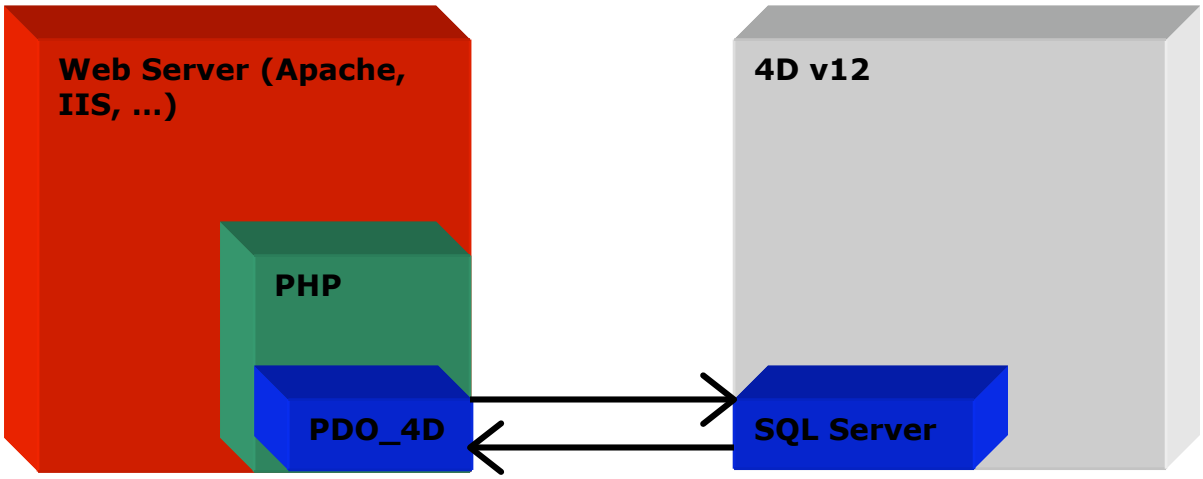
Note *The 4D database does not need to be run on the same machine as Apache web server.*

There are a number of different possible setups that match the above requirements. For example, the web servers that support PHP include Apache and Internet Information Server (IIS). Also, the operating systems that support Apache and/or IIS include Mac OS, Windows, and Linux. Some systems have one of these web server (and PHP) installed by default, but other systems require it to be manually installed. There are also third party applications that can run Apache and PHP, such as MAMP, XAMPP, WAMPServer. Which setup you choose is ultimately going to be dependant on your preferences and needs. The point here is that there are a variety of setups that you can choose from.

Setup (how does it work)

The web pages containing PHP code are hosted/served by the web server. In order for PHP code to be interpreted, the web server will need to have a PHP module installed. A PHP module can be pre-installed with the web server or it can be added manually.

Once a PHP module is installed, the PDO_4D extension can be added. At this point, PHP code can be written to include PDO commands that call on the PDO_4D driver, which communicates with the SQL Server within a 4D database.



Installation

This section details the installation process for one of the supported setups. This specific setup is not a recommendation or preferred setup to use. It is simply a common setup that is used in this Technical Note as an example, just to give you an idea of the specific steps needed to setup and use a PDO_4D driver.

Environment

Here are details/versions for the example setup:

- Machine: MacBook Pro
- Processor: Intel Core 2 Duo
- OS: Mac OS X, 10.5.8
- Web Server: Apache 2.2.11
- PHP: 5.2.8
- PDO_4D, built using version 0.2.1 of the source

Steps

Enable PHP within the Apache web server:

- open the file "/private/etc/apache2/httpd.conf" using a text editor and uncomment the line:

```
# LoadModule php5_module libexec/apache2/libphp5.so
```

  Meaning, change the line to be

```
LoadModule php5_module libexec/apache2/libphp5.so
```

- copy "/private/etc/php.ini.default" to "/private/etc/php.ini"

Next, install the PDO_4D driver:

- Open the file "/private/etc/php.ini" using a text editor and add the following line of code:

```
extension=pdo_4D.so
```

- Place the "PDO_4D.so" file in the PHP "extensions" folder.

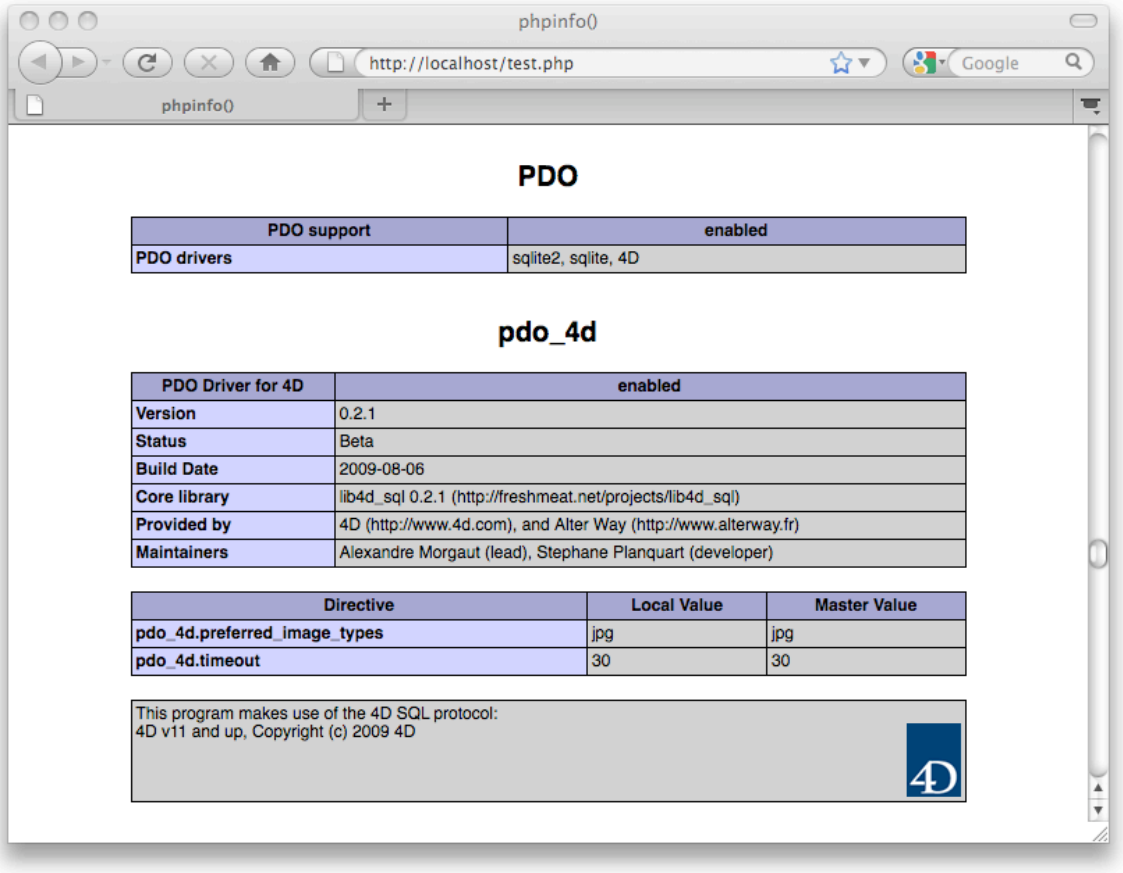
Note *This folder location is specified in the php.ini file, look for the line that starts with "extension_dir". For this setup, a new folder named "extensions" was created to keep all PHP extensions together. In order to use this new folder, the line was changed to be: extension_dir extension_dir = "private/etc/extensions".*

Lastly, start the apache web server: (There are 2 ways to accomplish this)

- In the Mac OS System Preferences, click on Sharing, and then check the "Web Sharing" option, or
- Open the Terminal and execute the following command:

```
sudo apachectl start
```

At this point the web server will be started with PHP enabled and with the PDO_4D driver installed. You are now able to serve PHP web pages that use the PDO_4D driver. If you have installed the PDO_4D driver successfully, you can validate the installation using the phpinfo() function (see Tip below for details), which should display the following information:



Tips/Things to watch out for

Before moving on to the examples, if this is the first time you are using an Apache web server, here are a few tips and things to look out for that can make the installation process run smoothly.

- Create a simple PHP web page that executes the "phpinfo()" function and that's it. The page can be as simple as the following:

```
<?php
   phpinfo();
?>
```

  When you open this page in a browser, it will display all the different information about the current PHP state. Meaning information about options, extensions, PHP version, web server, environment, paths to various files, …
- The following 3 Unix commands are very helpful for quickly starting and stopping the Apache server

```
sudo apachectl start
sudo apachectl stop
```

```
sudo apachectl restart
```

- Authentication is required when accessing many of the setup files mentioned above, even when logged in as an administrator. This is why you see the "sudo" in some of the Unix commands.
- Make sure logging errors to a file is turned on in the php.ini file. Here is the line to look for:

```
log_errors = on
```

  This will save all errors to the "/private/var/log/apache2/error_log" file, which is the default location. If you receive any unexpected errors, looking at this file is a good place to start.
- When creating web pages, make sure the files have the correct permissions. If the browser reports any permissions errors when opening a web page, look at the permissions of the files within the web root folder. An easy way to set all the permissions for the files within this directory is to execute the following Terminal command within the web root folder:

```
chmod 755 *.*
```

- PHP is case sensitive!!! $myvar and $myVar are not the same variable

Using PDO_4D

Once you have a web server and database setup, you can write the PHP code that accesses the database, which basically means executing SQL code and 4D methods.

Executing SQL code

Now that we have installed a PDO_4D driver, the next step is to setup a simple web page to access a 4D database using SQL code. Here are the steps:

- Create a new 4D database and start the SQL Server.
- Create a PHP file named test.php and place in the folder "/library/webserver/documents". This is the default web root location, which can be modified in the http.conf file. The line to modify should look like:

```
DocumentRoot "/Library/WebServer/Documents"
```

- Place the following PHP code into this test.php file:

```
<?php
	$dsn = '4D:host=localhost;port=19812;charset=UTF-8';
	$user = 'Administrator';
	$pswd = 'test';
	$db = new PDO($dsn, $user, $pswd);
```

```
	$db->exec('CREATE TABLE IF NOT EXISTS myTable(id INT NOT NULL, value
VARCHAR(100))');

	unset($db);

	echo 'done';  // if you see this then the code ran successfully
?>
```

- Go to the test page by opening a browser and entering the URL http://localhost/test.php. The resulting web page will simply display the word "done", but if you look at the 4D database, you will see that a table named "myTable" has been created.

Here is what is occurring in the above example:

```
$db = new PDO($dsn, $user, $pswd);
```

This line invokes the PDO_4D driver to create a new connection to the database. The information in the $dsn, $user, and $pswd variables specify the information that PHP will use to connect to the appropriate database. In this case, since we are using 4D as the database, the information in these variables will need to correspond to: the IP of the machine running the 4D database, the port that the SQL server is being published on, a valid username with a matching password for the 4D database.

```
$db->exec('CREATE TABLE IF NOT EXISTS myTable(id INT NOT NULL, value
VARCHAR(100))');
```

This line calls upon the PD_4D driver to execute the SQL code to create the "myTable" table.

Note *The 4D SQL engine is SQL-92 compliant, so the SQL code used must comply with this standard.*

```
unset($db);
```

Unset is a PHP command that is used to clean up variables.

```
echo 'done';  // if you see this then the code ran successfully
```

Echo is a PHP command that prints strings.

Executing 4D methods

4D methods can be executed using a specific SQL syntax. Here are the steps:

- Using the same database as above, add a method named "myAdd"
- Place the following code in this method:

```
C_LONGINT($0)
C_LONGINT($1;$2)

$0:= $1+$2
```

- For this method select the "Available through SQL" method property
- Create a PHP file named test2.php and place in the folder "/library/webserver/documents".
- Place the following PHP code into this test2.php file:

```
<?php
	$dsn = '4D:host=localhost;port=19812;charset=UTF-8';
	$user = 'Administrator';
	$pswd = 'test';

	$db = new PDO($dsn, $user, $pswd);

	$stmt = $db->prepare('SELECT {FN myAdd(1, 3) AS INT } FROM
_USER_SCHEMAS LIMIT 1');
	$stmt->execute();

	$results_array = $stmt->fetchAll();
	echo 'The result of the addition is: ' . $results_array[0][0] .
'<br>';

	unset($stmt);
	unset($db);
?>
```

- Go to the test page by opening a browser and entering the URL http://localhost/test2.php. The resulting web page will display the text "The result of the addition is: 4". Where the 4 is the value returned from the 4D method. While this is a trivial example, it shows how to call a 4D method, pass in parameters, and retrieve the returned value

First, notice that the connection code is the same as in the previous example. Now the method is executed by using a specific SQL syntax, however, when executing a method, the SQL code first needs to be prepared.

```
$stmt = $db->prepare('SELECT {FN myAdd(1, 3) AS VARCHAR } FROM _USER_SCHEMAS
LIMIT 1');
$stmt->execute();
```

So the `$db->prepare` command prepares the statement and this allows the `$stmt->execute()` command to execute the SQL code without having to specify the code in the call. In the above example, we are specifying 2 parameters for the myAdd method and specify that the return value be of type VARCHAR.

```
$results_array = $stmt->fetchAll();
echo 'The result of the addition is: ' . $results_array[0][0] . '<br>';
```

In the above code, the first line gets the returned value from the executed SQL code and stores the value in a 2 dimensional array. The next line then prints out the results of the method, meaning it extracts the returned value from with

2 dimensional array. The return format of `fetchAll()` and how to access the individual array items is detailed further in the CRUD example.

Example – CRUD

CRUD is an acronym that is often used to describe basic or essential database functionality. CRUD stands for Create, Read, Update, and Delete. Typically within the context of records: create records, read records, etc… In the included example, we have four PHP web pages that will implement each CRUD operation, and a few other operations, particularly creating and deleting tables.

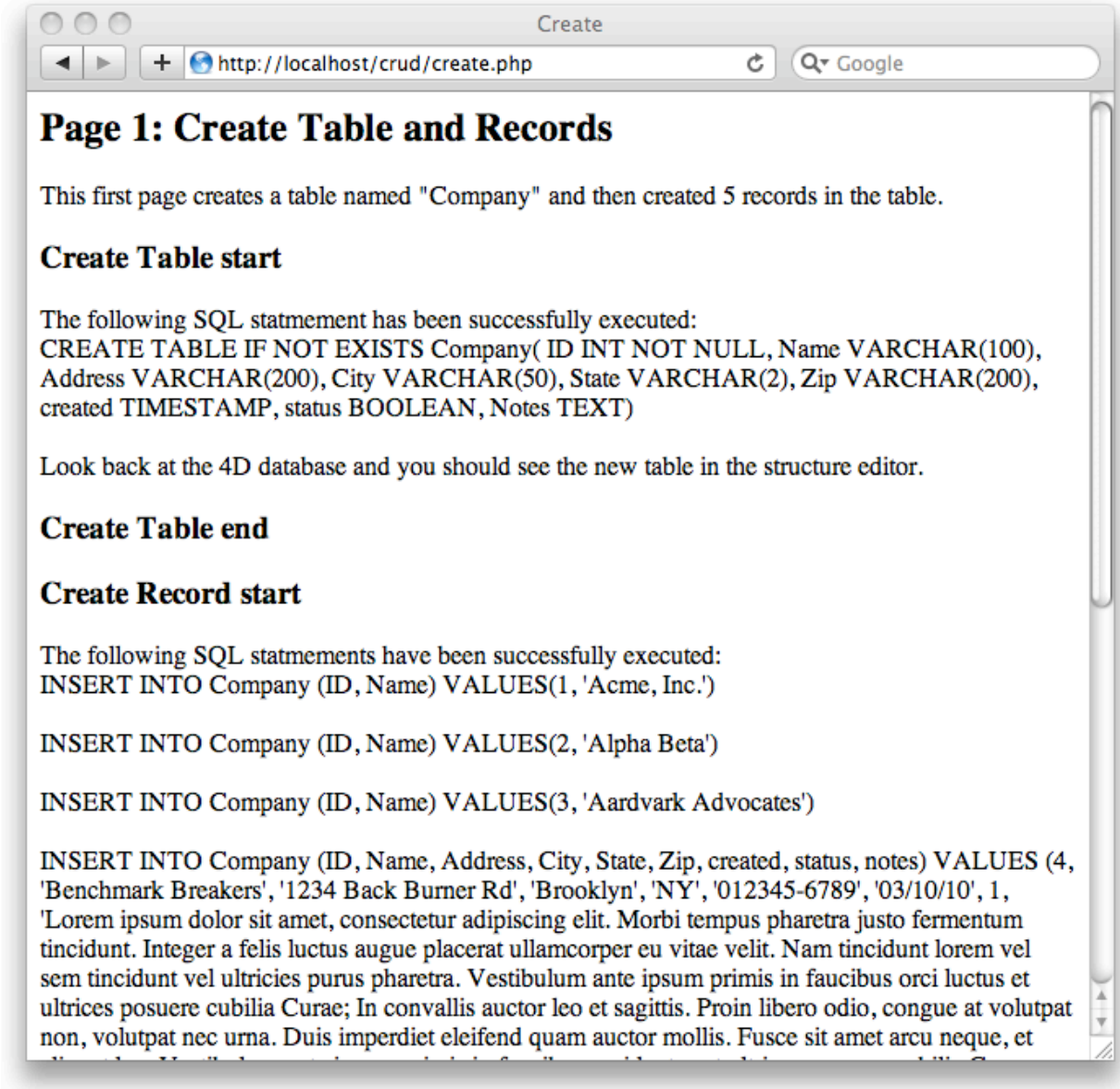
In order to use the example, follow these steps:

- Open the included database "CRUD.4dbase" with 4D v12 (the SQL server should automatically start)
- Setup a web server and install a PDO_4D driver (preferably on the same machine as the 4D database).
- Start the web server
- Place the included folder "Crud" to the web root folder of the web server. In the example above, that location would be "/library/webserver/documents"

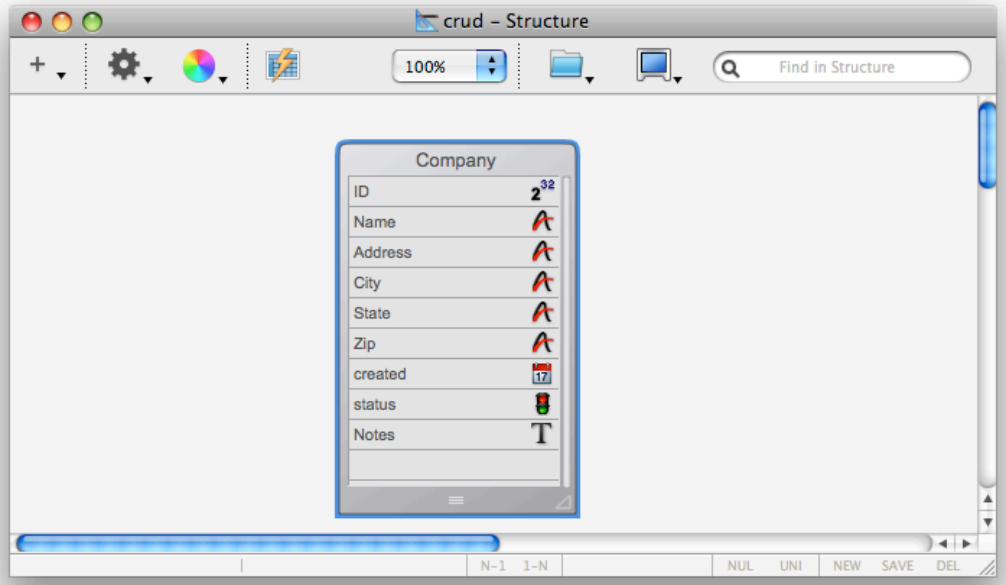
Note *The examples included in this Tech note were run in the environment mentioned in the "PDO_4D: Requirements, Setup, and Installation" section above.*

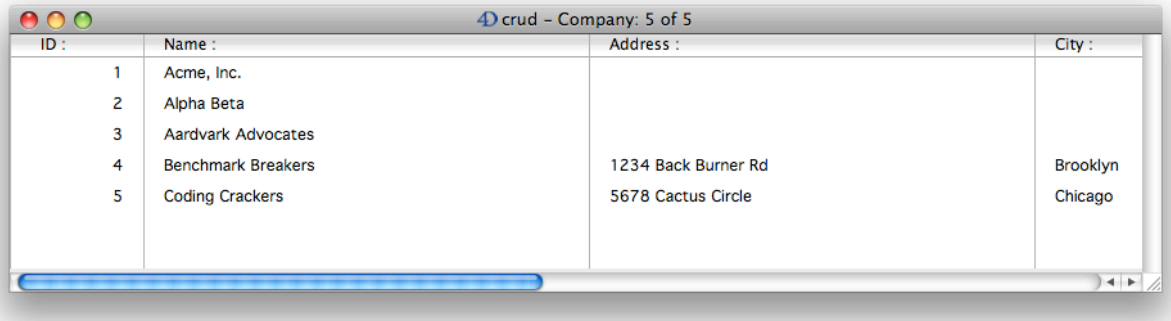
Create

To run this example, open a browser and enter the URL for the Create.php page. Assuming that you have opened the browser on the same machine as the web server and assuming that the web server is publishing on port 80, then the URL to input would be http://localhost/CRUD/Create.php. If everything has been setup correctly, then you should see a simple web page that looks like the following:



In this example, the actual content of the web page is not important, because at this point we only are concerned with the PHP code that is executed. When the page is brought up, it will create 1 table and 5 records in that table. After the page has been opened, you can validate that everything has successfully been created by looking back at the 4D database. You should see the following:





Here is a step by step explanation of the PHP code within this page:

```
$dsn = '4D:host=localhost;port=19812;charset=UTF-8';
$user = 'Administrator';
$pswd = 'test';

$db = new PDO($dsn, $user, $pswd);
```

This sets up the connection variables and makes the connection to the 4D SQL Server.

```
$create_stmt = 'CREATE TABLE IF NOT EXISTS Company( ' .
    'ID INT NOT NULL, ' .
    'Name VARCHAR(100), ' .
    'Address VARCHAR(200), ' .
    'City VARCHAR(50), ' .
    'State VARCHAR(2), ' .
    'Zip VARCHAR(200), ' .
    'created TIMESTAMP, ' .
    'status BOOLEAN, ' .
    'Notes TEXT)';

$db->exec($create_stmt);
```

This creates a table with the name "Company". The dot (.) is PHP syntax for string concatenation. So the first 10 lines here simply builds the SQL command and stores it as text in the variable $create_stmt. The next line actually executes the SQL command.

```
$id = 3;
$today_date = date("m/d/y");
$status = 1;
$notes = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi tempus
pharetra justo fermentum tincidunt. Integer a felis luctus augue placerat
ullamcorper eu vitae velit. Nam tincidunt ……";
```

These 4 lines setup the variables that are going to be used in the add record examples.

```
$add_stmt = "INSERT INTO Company (ID, Name) VALUES(1, 'Acme, Inc.')";
$db->exec($add_stmt);
```

This inserts 1 record in the "Company" table. Only 2 fields are populated in this line.

```
$add_stmt = "INSERT INTO Company (ID, Name) VALUES(" . 2 . ", 'Alpha Beta')";
$db->exec($add_stmt);
```

This inserts another record in the "Company" table, but uses a different PHP syntax for including the Integer value.

```
$add_stmt = "INSERT INTO Company (ID, Name) VALUES($id, 'Aardvark Advocates')";
$db->exec($add_stmt);
```

This also inserts a record in the "Company" table, but uses yet another PHP syntax for including the Integer value. In this case, PHP interprets the $id variable and places the value of the variable (3) into $add_stmt.

```
$add_stmt = "INSERT INTO Company (ID, Name, Address, City, State, Zip, created,
status, notes) " .
"VALUES (4, 'Benchmark Breakers', '1234 Back Burner Rd', 'Brooklyn', 'NY',
'012345-6789', '$today_date', $status, '$notes')";
$db->exec($add_stmt);
```

This inserts another record and populates all fields with values.

```
$add_stmt = "INSERT INTO Company " .
"VALUES (5, 'Coding Crackers', '5678 Cactus Circle', 'Chicago', 'IL', '98765-
4321', '$today_date', $status, '$notes')";
$db->exec($add_stmt);
```

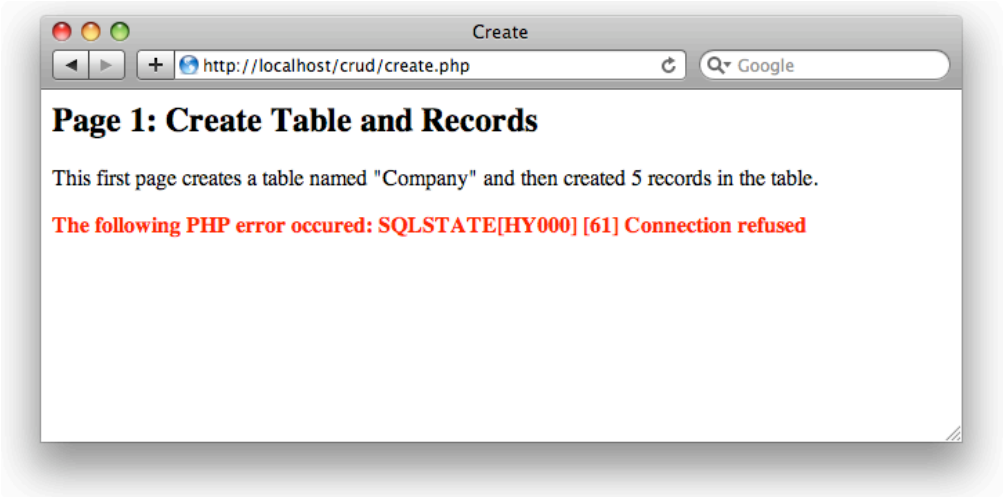
This also inserts a record with all fields populated, but does not include the column references.

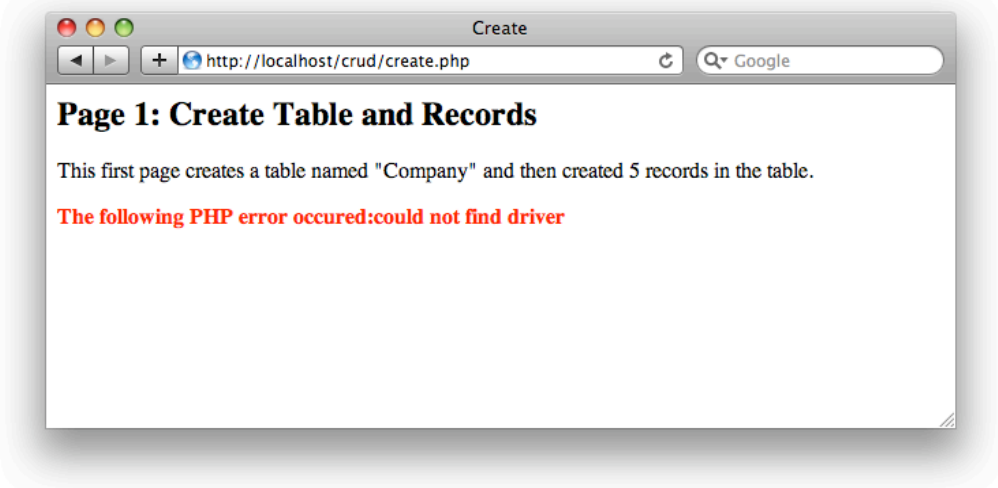
One last thing to mention is the inclusion of the try/catch block. This is something that is included in all 4 web pages in this example and it is used to handle errors that can possibly occur when using the PDO_4D driver. Here is the syntax that is used in these examples:

```
try {
 // do something
 // in this case, that means use PDO_4D to access a 4D database

} catch (Exception $e) {
   echo '<p style="color:red; font-weight:bold">' .
   'The following PHP error occured :' . $e->getMessage() .
   '</p>';
}
```

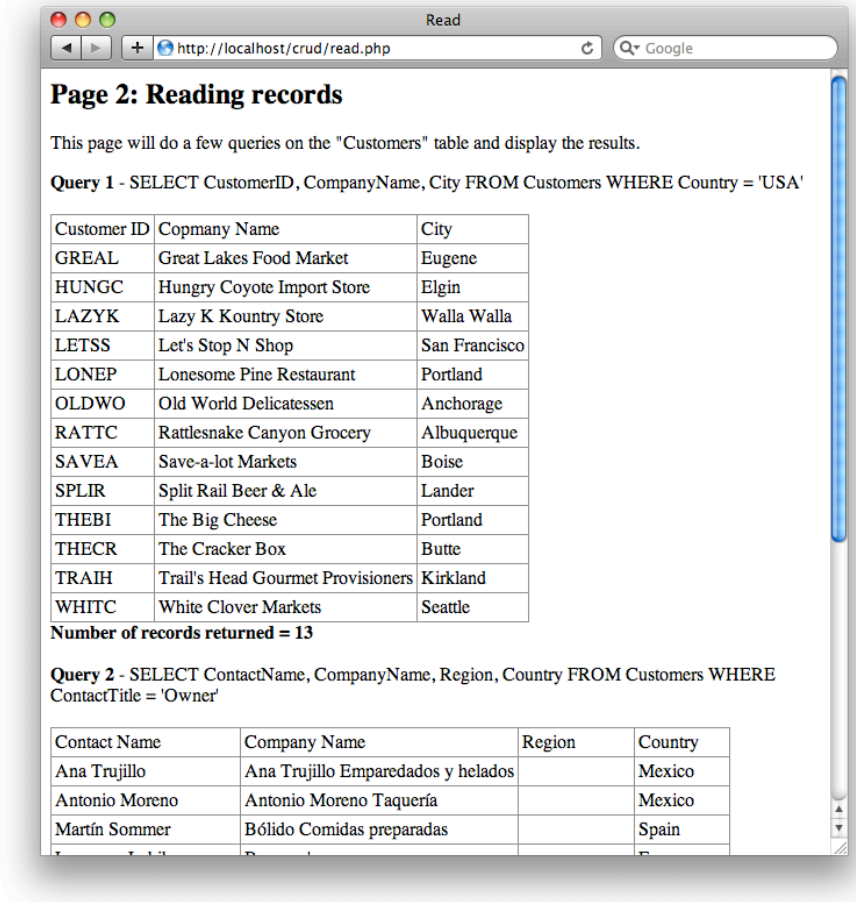
This will print a simple customized message out to the web page with the specific error message. For example, here are a few error messages that you can receive if you access the create.php page and either 1) the driver is not installed or 2) the SQL server is not started for the 4D database.





Read

To run this example, open a browser and enter the URL for the Read.php page. This page will run a query from the 4D database and display the results in the resulting web page. Upon opening this page, you should see the following:



Here is a explanation of the PHP code within this page:

```
$sql = 'SELECT CustomerID, CompanyName, City FROM Customers WHERE Country =
\'USA\' ';
$stmt = $db->prepare($sql);
$stmt->execute();
$results_array = $stmt->fetchAll();
```

This is a select statement that uses a hardcoded value in the WHERE clause. The statement is executed the same as in the Create example above, but this time, the line `$results_array = $stmt->fetchAll();` is added to get all of the records returned from the select statement. The resulting value in `$results_array` is an 2 dimensional array. Meaning it is an array with each element being another array, which contains the data from 1 record.

Arrays can have a variety of different constructions in PHP. To better clarify the structure of the 2 dimensional array returned from the `$stmt->fetchAll();` function, here is a graphical representation of two records returned from this function:

```
[0] => Array
    (
        [CUSTOMERID] => ALFKI
        [0] => ALFKI
        [COMPANYNAME] => Alfreds Futterkiste
        [1] => Alfreds Futterkiste
        [CITY] => Berlin
        [2] => Berlin
    )

[1] => Array
    (
        [CUSTOMERID] => ANATR
        [0] => ANATR
        [COMPANYNAME] => Ana Trujillo Emparedados y helados
        [1] => Ana Trujillo Emparedados y helados
        [CITY] => MÃ©xico D.F.
        [2] => MÃ©xico D.F.
    )
```

Next we will need to access the individual elements of the 2 dimensional array.

```
echo '<strong>Query 1</strong> - ' . $sql . '<br><br>';
echo '<table  BORDER=1 CELLPADDING=3 CELLSPACING=1 RULES=ALL FRAME=BOX">';
echo '<tr>' . '<td>Customer ID</td>' . '<td>Copmany Name</td>' .
'<td>City</td>' . '</tr>';
$index = 0;
foreach ($results_array as $id) {
```

```
    echo '<tr>';
    echo '<td>' . $results_array[$index]['CUSTOMERID'] . '</td>';
    echo '<td>' . $results_array[$index]['COMPANYNAME'] . '</td>';
    echo '<td>' . $results_array[$index]['CITY'] . '</td>';
    echo '</tr>';
    $index++;
}
echo '</table>';
echo '<br><br>';
```

The above code assembles the HTML code that basically builds a <table> that contains the returned records. `foreach` is used to iterate through the first level of arrays and then the individual columns are accessed using the [][] syntax and the correct indices. In the above example, we use a number as the index for the appropriate first level element (the array) and then the field name as the index for the second level (the actual column data). Looking back at the array structure above, we could have used a number for the second level. For example, in the above code, we could have achieved the same results with the following:

```
    …
    echo '<td>' . $results_array[$index][0] . '</td>';
    echo '<td>' . $results_array[$index][1] . '</td>';
    echo '<td>' . $results_array[$index][2] . '</td>';
    …
```

Next, we will get the count for the same query above.

```
$stmt = $db->prepare('SELECT COUNT(CustomerID) AS NumRecords FROM Customers
WHERE Country = \'USA\' ');
$stmt->execute();
$results_array = $stmt->fetchAll();
echo '<strong>Number of records returned = ' . $results_array[0][0] .
'</strong><br><br>';
```

Here, we process the query and use `$stmt->fetchAll();` to retrieve the results, but this time since we know that only 1 record is going to be returned and that the record will only have one column, we can use [0][0] to get the count.

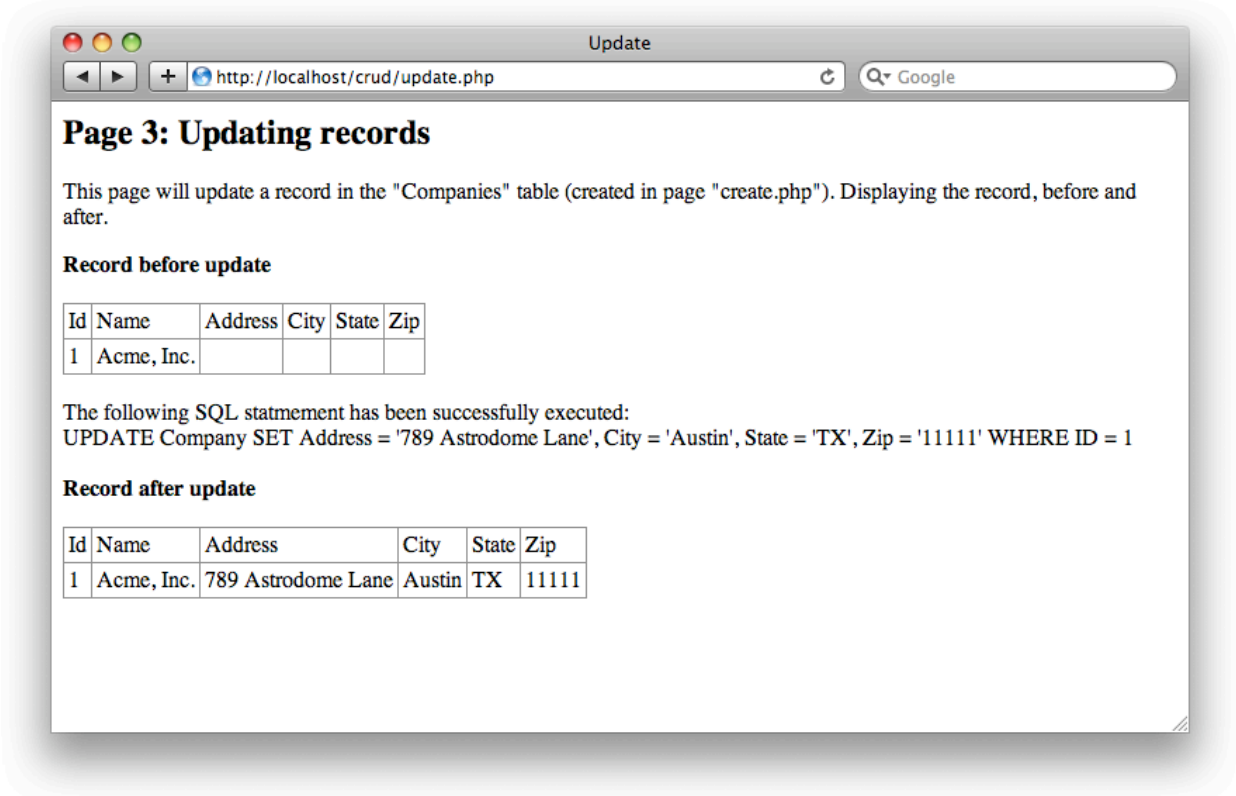
```
$title = "Owner";
$stmt = $db->prepare('SELECT ContactName, CompanyName, Region, Country FROM
Customers WHERE ContactTitle = \'' . $title . '\' ');
$stmt->execute();
$results_array = $stmt->fetchAll();
```

In the above code, we do another select statement, but this time we use a PHP variable in the WHERE clause instead of using a hardcoded value. The rest of the code is similar to the first example, just with different column names.

Update

To run this example, open a browser and enter the URL for the Update.php page. This page will update 1 record from the 4D database and display the

results in the results before and after the update was made. Upon opening this page, you should see the following:



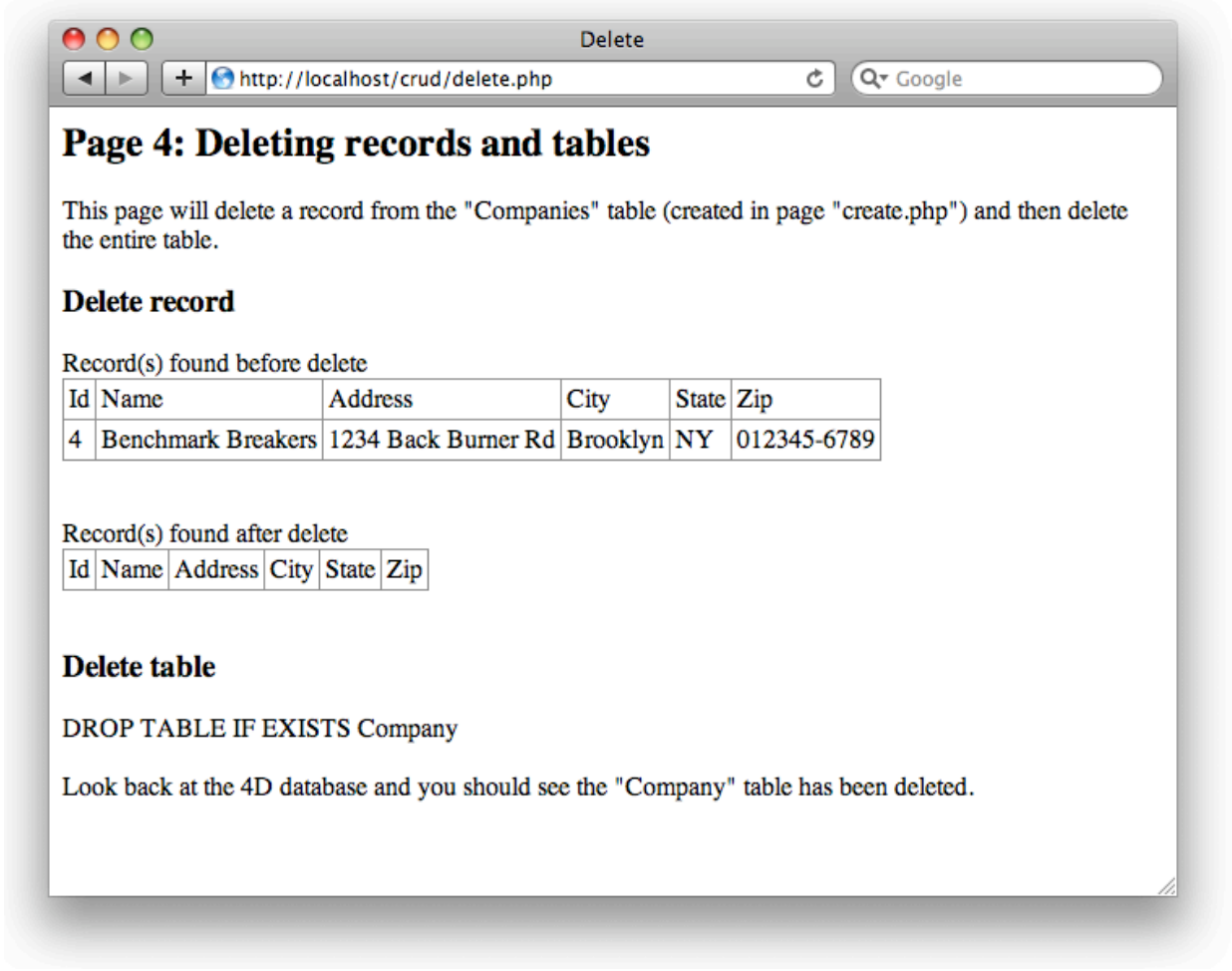
Most of the code in this page is similar to what was covered in the previous pages. The main new functionality in this page is in the following 3 lines:

```
$sql_update = 'UPDATE Company SET Address = \'789 Astrodome Lane\', City =
\'Austin\', State = \'TX\', Zip = \'11111\' WHERE ID = 1 ';
$stmt = $db->prepare($sql_update);
$stmt->execute();
```

This is the same basic setup to run SQL as we have seen above, just using the Update command now to change a few fields. One thing to note is the escaping of the single-quote character(`).

Delete

To run this example, open a browser and enter the URL for the Delete.php page. This page will update 1 record from the 4D database and display the results in the results before and after the update was made. Upon opening this page, you should see the following:



Most of the code in this page is similar to what was covered in the previous pages. The delete functionality in this page is in the following 3 lines:

```
$sql_delete = 'DELETE FROM Company WHERE ID = 4 ';
$stmt = $db->prepare($sql_delete);
$stmt->execute();
```

As in the update example, we just change the SQL statement and we are able to delete a record.

```
$sql_delete = 'DROP TABLE IF EXISTS Company ';
$stmt = $db->prepare($sql_delete);
$stmt->execute();
```

This last thing we do to finish the whole example is to delete the table. After executing the above code, the "Company" table will be deleted from the 4D database.

Conclusion

The ability to use 4D with 3rd party web servers such as Apache and IIS opens up new and exciting possibilities. This gives 4D developers more options when developing their solutions. Also by using on SQL code, it removes the requirement that web developers know 4D. This means that PHP experts can focus on building high quality web sites without needing to worry about the database side as much.

Now that you have an understanding of what the PDO_4D project is about, and how to use a PDO_4D driver, you have the skills necessary to use 4D as the backend to any 3rd party web server that supports PHP.

For more information

Here are some links to documentation for the different technologies discussed in this Tech Note:

http://pecl.php.net/package/PDO_4D/download - PDO_4D source download page

http://php.net/manual/ref.pdo-4d.php - PDO_4D documentation

http://php.net/ - documentation and examples for all things PHP.

http://www.apache.org/ - Apache Software Foundation page

http://www.iis.net/ - Microsoft Corporation IIS page