INFORMIX DBA
JEAN GEORGES PERRIN

PHP AND IDS

The popular Web-development language provides a powerful and easy way to access Informix data.

PHP (a recursive acronym for "PHP: Hypertext Preprocessor") has been around since 1995, but is often disregarded in favor of Java and .Net. However, its ease of use, natural integration with Linux, Apache, and the rest of the LAMP platform, and low price tag (free) make it a must for the fast-growing Web industry.

PHP has matured over the past decade, adding higher levels of developer services, including object orientation (OO), data objects (PDO), session handling, unit testing, integrated development environments (IDE), and debuggers.

IBM has released (as open source) the PDO driver for Informix IDS, and PHP 5 and its object structure are now mature and recognized by the industry.

In this column, I'll introduce PHP in the context of Informix. I'll show you how to check that your installation is up and running, and how to query the famous "stores" database. Other examples include inserting an element into the database, updating the element, and using transactions. Finally, I'll explain error handling in the context of PDO.

REQUIREMENTS

This article assumes you have a working Informix V10 and PHP V5.1 environment. We won't be using the configuration file for Informix or Apache; I'll show you what modifications are needed in php.ini, the PHP configuration file.

All examples run on a Windows XP box, IBM Informix IDS V10.0.TC4 and PHP V5.1.4. All examples will be command-line. Adding Apache http Server V2.0.55 creates our perfect WAIP platform. I'm running Eclipse V3.2RC6 with the Zend PHP plug-in V0.1.3.

CHECKING THE PHP V5 SIDE

Check that your php.ini file includes an extension entry for Informix. It might be commented (with a semicolon) but should be uncommented. You should see:

```
extension=php_pdo_informix.dll
```

To check that everything is working, let's try a simple "Hello world." Type the following in a hello.php file:

```
<?php
echo ('hello, Informix world');
?>
```

You can run it by entering:

```
C:\>php hello.php
hello, Informix world
```

This command provides a complete description of your PHP installation:

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

Scroll down (or grep on informix), and you'll see:

```
PDO drivers => informix
pdo_informix
pdo_informix support => enabled
```

Note that the phpinfo() function can be called within a Web page and displays a graphical result. Usage of grep is slightly more complex in a Web page.

CHECKING THE IDS SIDE

We'll be using the stores demo, which is a basic database describing a shop, with tables such as Customer, Articles, and so on.

If stores isn't installed on your server, here's how to install it:

```
dbaccessdemo -log -dbspace dbs_xeo
```

in which:

- -log says you wish to use transaction logging
- -dbspace dbs_xeo indicates the name of the dbspace you want to use (replace with one of your dbspaces or leave blank).

After a few seconds, the following message is returned:

```
{...}
Database closed.

The stores_demo database created
  successfully.
The demo files can be found in
  C:\PROGRA~1\IBM\Informix\demo\dbacces
  s\demo_ids
```

THE FIRST SELECT

Before we get started, here are a few PHP tips:

- Variables are always prefixed by a dollar sign and aren't declared.
- Strings can be either quoted or double quoted (In PHP, when a string is delimited by simple quotes, variables and escape sequences for special characters won't be expanded).
- All PHP scripts are enclosed in a <?php ... ?> block.
- Blocks are surrounded by curly braces (as in C and Java).
- Instructions are separated by semicolons (as in C and Java).

Let's try listing all the customers now. Listing 1 shows the code. In the listing, we first define a PDO object that takes three arguments: a connection string, a user, and a password. You can't guess the connection string, but it is fairly idiomatic. It starts with the driver you want to use (in this case, informix).

The next parameters are all parameter=value pairs, separated by semicolons (;). You need:

- host, the hostname (xeo)
- service, the service port (1526)
- database, the database you want to use (stores_demo)
- server, the data server (ol_xeo)
- protocol, tells the driver how to access the database (olsoctcp).

In addition, you can add EnableScrollableCursors=1 to enable scrollable cursors and specify the locale, which I've found useful.

PHP now connects to the engine. If the engine is down, this line will fail. Next, we ask the database ($db) to build a result set ($rs) from the query SELECT * FROM customer. We get the number of columns this result set returned.

In the section marked IV, for each row ($row) of the result set ($rs), we echo the column's value.

Easy and straightforward, right? Note that you can access the value of a row by its index, as shown in the $row[$i] call. If you don't force the carriage return (echo ("\n");), PHP will continue on the same line.

If you want a clipped result, use rtrim(), as in the following:

```
echo (rtrim($row[$i]) . "|");
```

If you want a digested result, you use md5():

```
echo (md5($row[$i]) . "|");
```

The result is a lot less human-readable, but in these service-oriented architecture (SOA) days, you might want to add some MD5 checksums.

An easy extension to this example is to add the column names on the very first line of the dump. PDO accesses the database's meta information like a charm. Before the foreach{...} loop, add:

```
for ($i = 0; $i < $colCount; $i++) {
  $meta = $rs->getColumnMeta($i);
  $echo ($meta['name'] . "|");
}
echo ("\n");
```

LISTING 1. select.php.

```
<?php
$i = 0;

$db = new PDO(
"informix:host=xeo;service=1526;database=stores_demo;server=ol_xeo;protocol=olsoctcp;EnableScrolla
bleCursors=1;DB_LOCALE=EN_US.8859-1"/* connection string */ ,
  "informix"/* user */ ,
  "informix"/* password */); // (I)

$rs = $db->query("SELECT * FROM customer"); // (II)
$colCount = $rs->columnCount(); // (III)

foreach ($rs as $row) { // (IV)
 for ($i = 0; $i < $colCount; $i++) {
  echo ($row[$i] . "|");
 }
 echo ("\n");
}
?>
```

LISTING 2. insert.php.

```
<?php
$db = new PDO(
"informix:host=xeo;service=1526;database=stores_demo;server=ol_xeo;protocol=olsoctcp;EnableScrolla
bleCursors=1;DB_LOCALE=EN_US.8859-1", "informix", "informix");

$sqlstmt = 'insert into customer(customer_num, fname, lname, company) values(?, ?, ?, ?);';
$stmt = $db->prepare($sqlstmt); // (I)
if ($stmt == false) {
 echo ("Error while preparing the statement.\n");
 die(); // (III)
}

$dataarray = array('0', 'Jean Georges', 'Perrin', 'None'); // (II)
$result = $stmt->execute($dataarray);
if ($result == true) {
 echo ("Row successfully insterted.\n");
}
else {
 echo ("An error occured while inserting a row.\n");
}
?>
```

INSERT A ROW

Inserting a row with PHP is as easy as in any other language. Listing 2 shows how to use a prepared statement to insert values in the database using arrays. The connection string is the same as for the SELECT, but the SQL statement, of course, is different. In this example, we're asking the database to prepare it (see the section marked I). Note that the statement variable ($stmt) can be considered as a boolean value or as an instance of an object. This "weak typing" reminds me of 4GL.

After preparation, an array is built using the array() function and passed to the execute() method of the statement (see the section marked II.) The die() function tells the application to die (III).

UPDATE THE COMPANY

In our earlier INSERT example, we entered a record for Jean Georges Perrin without a company name. Let's update my personal record to enter a company name.

The example in Listing 3 includes support for transactions (see page 60). Transactions can be stated by the beginTransaction() method (see I) and rolled back or committed by the rollBack() (see II) and commit() (see III) methods.

In this example, we're not preparing the statement. We're simply using the exec() method, which expects a simple SQL statement.

HANDLING ERRORS

There are two kinds of errors: predictable, such as the test (if ($result == false) {..}) in Listing 3; and unpredictable, such as the database ceasing to respond. The latter are processed through exceptions.

As in Java, PHP V5 sends exceptions when a problem occurs. In Java, you must catch or throw all exceptions. In PHP, there's no requirement to catch or throw them.

Listing 4 shows how to catch and process an exception on the spot. Listing 5 shows how an exception climbs the function stack and is processed later. In both cases, the message you'll get will be something like:

```
SQLSTATE=28000, SQLDriverConnect: -951
  [Informix][Informix ODBC Driver][Informix]
  Incorrect password or user scott@XEO is
  not known on the database server.
```

-951 is the Informix error, and 28000 is the ANSI SQLSTATE error that matches Authorization name is invalid.

LISTING 3. update.php.

```
<?php
$db = new PDO(
"informix:host=xeo;service=1526;database=stores_demo;server=ol_xeo;protocol=olsoctcp;EnableScrolla
bleCursors=1;DB_LOCALE=EN_US.8859-1", "informix", "informix");

$db->beginTransaction(); // (I)
$sqlstmt = "update customer set company='IBM' where fname='Jean Georges' and lname='Perrin'";
$result = $db->exec($sqlstmt);
if ($result == false) {
  echo ("Error while updating the database.\n");
  $db->rollBack(); // (II)
  die();
}
$db->commit(); // (III)
?>
```

LISTING 4. Exception caught and processed on the spot.

```
<?php
try {
  $db = new PDO(
"informix:host=xeo;service=1526;database=stores_demo;server=ol_xeo;protocol=olsoctcp;EnableScrolla
bleCursors=1;DB_LOCALE=EN_US.8859-1"/* connection string */,
    "scott"/* user */,
    "tiger"/* password */);
}
catch (PDOException $e) {
  echo ("Could not connect to Informix server:\n" . $e->getMessage() . "\n");
}
?>
```

LISTING 5. Exception climbing the function stack.

```
<?php
function listCustomers() {
  try {
    $db = dbConnect();
  }
  catch (PDOException $e) {
    echo ("Exception processed here: " . $e->getMessage() . "\n");
  }
  dbSelect($db);
}

function dbConnect() {
  $db = new PDO(
"informix:host=xeo;service=1526;database=stores_demo;server=ol_xeo;protocol=olsoctcp;EnableScrolla
bleCursors=1;DB_LOCALE=EN_US.8859-1"/* connection string */,
    "scott"/* user */,
    "tiger"/* password */);
  return $db;
}

function dbSelect($db) {
  // Select statement code.
}

listCustomers();
?>
```

PHP'S PLACE

PHP is powerful, easy to learn and very hip with developers right now. New object-orientation capabilities make it appropriate for larger-scale projects, yet it remains an easy scripting language for batch or small test apps.

PHP is appropriate in SOA and in integration projects because it is lighter to implement than Java, is widely available, and offers many other benefits. PDO adds a level of abstraction which makes business applications easier to build than with previous versions of PHP.

For help exploring PHP in an Informix environment, visit the development tools forum on the International Informix User Group Web site. ■

Jean Georges Perrin [jgp@iiug.org] has been involved in software engineering for the last decade using development languages such as Informix-4GL, Java, Visual Basic, C, C++, and PHP. He has served on IIUG Board of Directors since 2002.

RESOURCES

PHP.net
www.php.net

PDO
www.php.net/pdo

Informix PDO
www.php.net/manual/en/ref.pdo-informix.php

ANSI SQL 1992
www.andrew.cmu.edu/user/shadow/sql/sql1992.txt

IIUG Development Tools Forum
www.iiug.org/forums/development-tools