PHP Data Objects

Wez Furlong
<wez@omniti.com>

About the author

- PHP Core Developer since 2001
- Author of the Streams layer
- “King” of PECL
- Author of most of PDO and its drivers
- Day-job is developing the Fastest MTA on Earth

Coming up…

- Problems with PHP DB access
- PDO Solution
- Features / Drivers
- Installation
- Getting Started
- Features

The Problem

- No consistency of API between DB extensions
- Sometimes no self-consistency within a given extension
- Duplicated code (but not)
- High maintenance

The PDO Solution

- Move PHP specific stuff into one extension
- Database specific stuff (only) in their own extensions
- Data access abstraction, not database abstraction

Features

Performance

- Native C code beats a scripted solution
- Takes advantage of latest PHP 5 internals

Features

Power

- Gives you common DB features as a base
- Still be able to access specialist functions

What can it do?

- Prepare/execute, bound parameters
- Transactions
- LOBS
- Scrollable Cursors
- SQLSTATE error codes, flexible error handling
- Portability attributes

Available Drivers

- Pretty much all of them:
- MySQL, PostgreSQL
- ODBC, DB2, OCI
- SQLite (2 and 3)
- Sybase/FreeTDS/MSSQL
- Firebird (needs love)

Installing

- Ships with PHP 5.1
- Available for PHP 5.0.x via PECL
- DLLs downloadable for both on Windows via http://pecl4win.php.net

Building w/ PHP 5.0

```
./configure \
        --with-zlib \
        --prefix=/usr/local/php5
```

- `pear upgrade pear`
- `pear install PDO`
- `extension=pdo.so` in your php.ini

Building w/ PHP 5.0

- Select the driver(s) you need
- `pear install PDO_XXX`
- `extension=pdo_xxx.so` in your php.ini

Installing on Win32

- Grab the DLLs from the snaps site http://pecl4win.php.net/
- You need:
  - php_pdo.dll
  - php_pdo_XXX.dll
- Put them in C:\php5\ext

Switching it on

- Need to enable PDO in your php.ini
- MUST load PDO first
- Unix:
  - `extension=pdo.so`
  - `extension=pdo_XXX.so`
- Windows:
  - `extension=php_pdo.dll`
  - `extension=php_pdo_XXX.dll`

Connecting via PDO

```
try {
   $dbh = new PDO($dsn, $user,
            $password, $options);
} catch (PDOException $e) {
   echo ”Failed to connect:”
            . $e->getMessage();
}
```

DSNs

- In general

  drivername:<driver-specific-stuff>

DSNs

- mysql:host=name;dbname=dbname
- pgsql:native_pgsql_connection_string

- odbc:odbc_dsn
- oci:dbname=dbname;charset=charset

DSNs

- sqlite:/path/to/db/file
- sqlite::memory:

- sqlite2:/path/to/sqlite2/file
- sqlite2::memory:

DSN Aliasing

- uri:uri
  - Specify location of a file containing actual DSN on the first line
  - Works with streams interface, so remote URLs can work too
- name (with no colon)
  - Maps to pdo.dsn.name in your php.ini
  - pdo.dsn.name=sqlite:/path/to/name.db

DSN Aliasing

- `pdo.dsn.name=sqlite:/path/to/name.db`

```
$dbh = new PDO("name");
```

Same as:

```
$dbh = new PDO("sqlite:/path/to/name.db");
```

Connection Management

```
try {
   $dbh = new PDO($dsn, $user, $pw);
   // use the database here
   // ...
   // done; release the connection
   $dbh = null;
} catch (PDOException $e) {
   echo ”connect failed:”
         . $e->getMessage();
}
```

Persistent PDO

- Connection stays alive between requests

```
$dbh = new PDO($dsn, $user, $pass,
 array(
  PDO::ATTR_PERSISTENT => true
 )
);
```

Persistent PDO

- Can specify your own cache key

```
$dbh = new PDO($dsn, $user, $pass,
 array(
  PDO::ATTR_PERSISTENT => “my-key”
 )
);
```

- Useful for keeping separate persistent connections

Persistent PDO

- The ODBC driver runs with connection pooling enabled by default
- “better” than PHP-level persistence
  - Pool is shared at the process level
- Can be forced off by setting
  - `pdo_odbc.connection_pooling=off`

Let’s get data

```
$dbh = new PDO($dsn);
$stmt = $dbh->prepare(
                    "SELECT * FROM FOO");
$stmt->execute();
while ($row = $stmt->fetch()) {
    print_r($row);
}
$stmt = null;
```

Forward-only cursors

- Also known as "unbuffered" queries in mysql parlance
- They are the default cursor type
- rowCount() doesn't have meaning
- FAST!

Forward-only cursors

- Other queries are likely to block
- You must fetch all remaining data before launching another query
- `$stmt->closeCursor()`

Buffered Queries

```
$dbh = new PDO($dsn);
$stmt = $dbh->query(
                    "SELECT * FROM FOO");
$rows = $stmt->fetchAll();
$count = count($rows);
foreach ($rows as $row) {
    print_r($row);
}
$stmt = null;
```

Data typing

- Very loose
- uses strings for data
- Gives you more control over data conversion
- Supports integers where 1:1 mapping exists
- Is float agnostic
  - PDO is precise

Fetch modes

- $stmt->fetch(PDO::FETCH_BOTH)
  - Array with numeric and string keys
  - default option
- PDO::FETCH_NUM
  - Numeric only
- PDO::FETCH_ASSOC
  - String only

Fetch modes

- PDO::FETCH_OBJ
  - stdClass object
  - $obj->name == ‘name’ column
- PDO::FETCH_CLASS
  - You choose the class
- PDO::FETCH_INTO
  - You provide the object

Fetch modes

- PDO::FETCH_COLUMN
  - Fetches only a particular column
- PDO::FETCH_BOUND
  - Only fetches into bound variables
- PDO::FETCH_FUNC
  - Returns the result of a callback
- *and more…*

Iterators

```
$dbh = new PDO($dsn);
$stmt = $dbh->query(
                "SELECT name FROM FOO",
                PDO::FETCH_COLUMN, 0);
foreach ($stmt as $name) {
     echo "Name: $name\n";
}
$stmt = null;
```

Changing data

```
$deleted = $dbh->exec(
    "DELETE FROM FOO WHERE 1");

$changes = $dbh->exec(
    "UPDATE FOO SET active=1 "
  . "WHERE NAME LIKE '%joe%'");
```

Autonumber/sequences

```
$dbh->exec(
     "insert into foo values (...)");
echo $dbh->lastInsertId();

$dbh->exec(
     "insert into foo values (...)");
echo $dbh->lastInsertId("seqname");
```

It's up to you to call it appropriately

Smarter queries

- Quoting is annoying, but essential
- PDO offers a better way

```
$stmt = $db->prepare("INSERT INTO CREDITS "
          ."(extension, name) "
          ."VALUES (:extension, :name)");
$stmt->execute(array(
    'extension' => 'xdebug',
    'name'      => 'Derick Rethans'
));
```

Smarter queries

- Quoting is annoying, but essential
- PDO offers a better way

```
$stmt = $db->prepare("INSERT INTO CREDITS "
          ."(extension, name) "
          ."VALUES (?, ?)");
$stmt->execute(array(
                    'xdebug',
                    'Derick Rethans'
));
```

$db->quote()

- If you still need to quote things “by-hand”
- *Currently a no-op for the ODBC driver*

Stored Procedures

```
$stmt = $dbh->prepare(
            "CALL sp_set_string(?)");
$stmt->bindParam(1, $str);
$str = 'foo';
$stmt->execute();
```

OUT parameters

```
$stmt = $dbh->prepare(
            "CALL sp_get_string(?)");
$stmt->bindParam(1, $ret,
                 PDO::PARAM_STR, 4000);
if ($stmt->execute()) {
    echo "Got $ret\n";
}
```

IN/OUT parameters

```
$stmt = $dbh->prepare(
                  "call @sp_inout(?)");
$val = "My Input Data";
$stmt->bindParam(1, $val,
                   PDO::PARAM_STR|
                   PDO::PARAM_INPUT_OUTPUT,
                   4000);
if ($stmt->execute()) {
    echo "Got $val\n";
}
```

Binding columns for output

```
$stmt = $dbh->prepare(
     "SELECT extension, name from CREDITS");
if ($stmt->execute()) {
 $stmt->bindColumn('extension', $extension);
 $stmt->bindColumn('name',      $name);
 while ($stmt->fetch(PDO::FETCH_BOUND)) {
    echo "Extension: $extension\n";
    echo "Author: $name\n";
 }
}
```

Portability Aids

- PDO aims to make it easier to write db independent apps
- Number of hacks^Wtweaks for this purpose

Oracle style NULLs

- Oracle translates empty strings into NULLs
- `$dbh->setAttribute(PDO::ATTR_ORACLE_NULLS, true);`
- Translates empty strings into NULLs when fetching data
- *But won't change them on insert*

Case folding

- The ANSI SQL standard says that column names are returned in upper case
- High end databases (eg: Oracle and DB2) respect this
- Most others don't

```
$dbh->setAttribute(
  PDO::ATTR_CASE,
  PDO::CASE_UPPER);
```

Error Handling

- PDO offers 3 different error modes
- $dbh->setAttribute(PDO_ATTR_ERRMODE, $mode);
  - PDO::ERRMODE_SILENT
  - PDO::ERRMODE_WARNING
  - PDO::ERRMODE_EXCEPTION
- Attempts to map native codes to SQLSTATE standard codes
- But still offers native info too

PDO::ERRMODE_SILENT

```
if (!$dbh->query($sql)) {
    echo $dbh->errorCode() . "<br>";
    $info = $dbh->errorInfo();
    // $info[0] == $dbh->errorCode()
    //      SQLSTATE error code
    // $info[1] is the driver specific
    //      error code
    // $info[2] is the driver specific
    //      error string
}
```

PDO::ERRMODE_WARNING

- Same as PDO::ERRMODE_SILENT
- But, raises an E_WARNING as errors are detected
- You can selectively suppress the warnings with @ operator

PDO_ERRMODE_EXCEPTION

```
try {
    $dbh->exec($sql);
} catch (PDOException $e) {
    // display warning message print
    $e->getMessage();
    $info = $e->errorInfo;
    // $info[0] == $e->code;
    //      SQLSTATE error code
    // $info[1] is the driver specific error code
    // $info[2] is the driver specific error string
}
```

Transactions

```
$dbh->beginTransaction();
try {
    $dbh->query("UPDATE ...");
    $dbh->query("UPDATE ...");
    $dbh->commit();
} catch (Exception $e) {
    $dbh->rollBack();
}
```

LOBs via Streams

- Large objects are usually 4kb or more in size
- Advantageous to avoid fetching them until you really need to
- Mature RDBMS offer LOB APIs for this
- PDO exposes LOBS as Streams

Fetching an image

```
$db = new PDO($dsn);
$stmt = $db->prepare(
     “select contenttype, imagedata”
    .“ from images where id=?”);
$stmt->execute(array($_GET['id']));
$stmt->bindColumn(1, $type, PDO::PARAM_STR,
  256);
$stmt->bindColumn(2, $lob, PDO::PARAM_LOB);
$stmt->fetch(PDO::FETCH_BOUND);
header("Content-Type: $type");
fpassthru($lob);
```

Inserting an image

```
$db = new PDO($dsn);
$stmt = $db->prepare(
            “insert into images ”
           .“(id, contenttype, imagedata) ”
           .“values (?, ?, ?)”);
$id = get_new_id();
$fp = fopen($_FILES['file']['tmp_name'],
            'rb');
$stmt->bindParam(1, $id);
$stmt->bindParam(2,$_FILES['file']['type']);
$stmt->bindParam(3, $fp, PDO::PARAM_LOB);
$stmt->execute();
```

Scrollable Cursors

- Allow random access to a rowset
- Higher resource usage than forward-only cursors
- Can be used to emulate limit, offset style paged queries
- Can be used for positioned updates (more useful for CLI/GUI apps)

Positioned updates

- An open (scrollable) cursor can be used to target a row for another query
- Name your cursor by setting PDO::ATTR_CURSOR_NAME during prepare()
- `UPDATE foo set bar = ? WHERE CURRENT OF cursor_name`

Multi-rowset queries

```
$stmt = $dbh->query(
      "call sp_multi_results()");
do {
   while($row = $stmt->fetch()) {
     print_r($row);
   }
} while ($stmt->nextRowset());
```

Questions?

Resources

- The PHP Manual: http://php.net/pdo
  - http://docs.php.net/pdo
- Gold:
  - http://troels.arvin.dk/db/rdbms/#select-limit-offset
- Publications
  - www-128.ibm.com/developerworks/db2/library/techarticle/dm-0505furlong/
  - www.oracle.com/technology/pub/articles/php_experts/otn_pdo_oracle5.html
- My blog: http://netevil.org
- Bugs: http://bugs.php.net