PHP Data Objects

Wez Furlong
<wez@messagesystems.com>

About me

- PHP Core Developer since 2001
- Author of the Streams layer
- I hold the title “King” of PECL
- Author of most of PDO and its drivers

What is PDO?

- PHP Data Objects
- A set of PHP extensions that provide a core PDO class and database specific drivers
- Focus on data access abstraction rather than database abstraction

What can it do?

- Prepare/execute, bound parameters
- Transactions
- LOBS
- SQLSTATE standard error codes, flexible error handling
- Portability attributes to smooth over database specific nuances

What databases are supported?

- MySQL, PostgreSQL
- ODBC, DB2, OCI
- SQLite
- Sybase/FreeTDS/MSSQL

Connecting

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

DSNs

- mysql:host=name;dbname=dbname
- pgsql:host=name dbname=dbname
- odbc:odbc_dsn
- oci:dbname=dbname;charset=charset
- sqlite:/path/to/file

Connection Management

```
try {

  $dbh = new PDO($dsn, $user, $pw);
  // use the database here
  // ...
  // done; release
  $dbh = null;

} catch (PDOException $e) {

  die($e->getMessage();

}
```

DSN Aliasing

- uri:uri
  - Specify location of a file that contains the actual DSN on the first line
  - Works with the streams interface, so remote URLs can work too (this has performance implications)
- 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”);

is equivalent to:

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

Persistent Connections

```
// Connection stays alive between requests

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

Persistent Connections

```
// 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
```

(requires that your web server be restarted)

Error Handling

- Maps error codes to ANSI SQLSTATE (5 character text string)
  - also provides the native db error information
- Three error handling strategies
  - silent (default)
  - warning
  - exception

PDO::ERRMODE_SILENT

```
// The default mode

if (!dbh->query($sql)) {
  echo $dbh->errorCode(), “<br>”;
  $info = $dbh->errorInfo();
  // $info[0] == $dbh->errorCode()
  //             SQLSTATE error code
  // $info[1] is driver specific err code
  // $info[2] is driver specific
  //             error message
}
```

PDO::ERRMODE_WARNING

```
$dbh->setAttribute(PDO::ATTR_ERRMODE,
                   PDO::ERRMODE_WARNING);
```

Behaves the same as silent mode

Raises an E_WARNING as errors are detected

Can suppress with @ operator as usual

PDO::ERRMODE_EXCEPTION

```
$dbh->setAttribute(PDO::ATTR_ERRMODE,
                   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] driver specific error code
  // $info[2] driver specific error string
}
```

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

- a.k.a. “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);
}
```

Data typing

- Very loose
- Prefers strings
- Gives you more control over data conversion

Fetch modes

- $stmt->fetch(PDO::FETCH_BOTH);
  - Array with numeric and string keys
  - default option
- PDO::FETCH_NUM
  - numeric keys only
- PDO::FETCH_ASSOC
  - string keys 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 a column (example later)
- PDO::FETCH_BOUND
  - Only fetches into bound variables
- PDO::FETCH_FUNC
  - Returns the result filtered through a callback
- *see the manual for 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”);
```

Its up to you to call the right one for your db!

Prepared Statements

```
// No need to manually quote data here

$stmt = $dbh->prepare(
   “INSERT INTO CREDITS (extension, name)”
  .“VALUES (:extension, :name)”);

$stmt->execute(array(
   ‘extension’ => ‘xdebug’,
   ‘name’      => ‘Derick Rethans’
));
```

Prepared Statements

```
// No need to manually quote data here

$stmt = $dbh->prepare(
   “INSERT INTO CREDITS (extension, name)”
  .“VALUES (?, ?)”);

$stmt->execute(array(
                  ‘xdebug’,
                  ‘Derick Rethans’
));
```

$db->quote()

- If you really must quote things “by-hand”
- $db->quote() adds quotes and proper escaping as needed
- But doesn’t do anything in the ODBC driver!
- Best to use prepared statements

Transactions

```
$dbh->beginTransaction();
try {
  $dbh->query(“UPDATE ...”);
  $dbh->query(“UPDATE ...”);
  $dbh->commit();
} catch (PDOException $e) {
  $dbh->rollBack();
}
```

Stored Procedures

```
$stmt = $dbh->prepare(
              “CALL sp_set_string(?)”);
$stmt->execute(array(‘foo’));



$stmt = $dbh->prepare(
              “CALL sp_set_string(?)”);

$stmt->bindValue(1, ‘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”;
}
```

Multi-rowset queries

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

Binding columns

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

Portability Aids

- PDO aims to make it easier to write db independent apps
- A 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);`

LOBs

- Large objects are usually >4kb in size
- Nice to avoid fetching them until you need them
- Mature RDBMS offer LOB APIs for this
- PDO exposes LOBs as Streams

Fetching an image

```
$stmt = $dbh->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);
```

Uploading an image

```
$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 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

Questions?

- Find these slides on my blog and on slideshare.net
- My blog: http://netevil.org/
- Gold: http://troels.arvin.dk/db/rdbms/#select-limit-offset