# Facultad de Ingeniería – UNMdP

# PHP y MySQL

## Programación de Páginas Web Dinámicas

**Segunda Parte**

**Versión 2.2**

**Coordinación:** **Carlos Rico**
**Autores:** **Leonardo Tadei**

Marzo 2011

# PDO – PHP Data Objects

## Introducción

La extensión PHP Data Objects (PDO) define un interfaz ligera y Orientada a Objetos, para tener acceso a bases de datos en PHP. Cada controlador de base de datos que implementa la interfaz PDO puede exponer base de datos específicas como funciones de extensión regular. Tenga en cuenta que no puede realizar las funciones de base de datos utilizando la extensión PDO por sí mismo, debe utilizar un controlador PDO de base de datos específica para tener acceso a un servidor de base de datos.

PDO proporciona una capa de abstracción acceso a datos, que significa que, independientemente de la base de datos que está utilizando, se utiliza las mismas funciones para realizar consultas y obtener datos. PDO no proporciona una abstracción base de datos; esto no reescribe SQL o emular características faltantes. Debe usar una capa de abstracción en toda regla, si necesita esto.

PDO con PHP 5.1, está disponible como una extensión PECL para PHP 5.0; PDO requiere las características nuevas de OO en el núcleo de PHP 5, y así no correr con versiones anteriores de PHP.

## Conexiones y administración de la Conexión

Las conexiones con el RDBMS se establecen al crear una instancia de la clase base PDO. No importa qué driver de conexión a la DB se vaya a usar, siempre se usará la clase PDO. El constructor acepta parámetros (mensajes) para especificar la fuente de datos (conocida como DSN) y opcionalmente el nombre de usuario y contraseña.

```php
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
?>
```

De haber un error en la conexión, una excepción de tipo PDOException se lanzará. Se pueden atrapar las excepciones para manejar la condición de error, o puede dejarse para ser manejada por un manejador de excepciones globales de la aplicación si se lo determina vía set_exception_handler().

```php
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
    foreach($dbh->query('SELECT * from FOO') as $row) {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
```

```
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Una vez realizada satisfactoriamente la conexión a la Base de Datos, se obtiene una instancia de la clase PDO, en estos ejemplos llamada $dbh. Para cerrar la conexión debe destruirse el objeto asegurándose de que todas las referencias el desaparezcan (garbage collection); puede hacerse esto asignado NULL a la variable que representa al Objeto. Si esto no se hace de forma explícita, PHP lo hará automáticamente cuando el script finalice.

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
// use the connection here


// and now we're done; close it
$dbh = null;
?>
```

## Las transacciones

Ahora que estamos conectados a través de PDO, se debe entender como PDO gestiona las transacciones antes de comenzar a emitir las consultas. Si usted nunca ha trabajado con transacciones antes, debe saber que con ellas se implementan las cuatro características principales de un RDBMS: Atomicidad, Consistencia, Aislamiento y Durabilidad (ACID). En términos sencillos, cualquier trabajo realizado en una transacción, incluso si se lleva a cabo en etapas, se garantiza que se aplicará a la base de datos de forma segura y sin interferencias de otras conexiones. El trabajo transaccional también se puede deshacer automáticamente a petición del cliente (siempre y cuando no lo ha completado), lo que hace que el control de errores en los scripts más fácil.

Las transacciones son generalmente aplicadas por "salvar" una serie de cambios que se aplicarán a la vez, esto tiene el buen efecto secundario de mejorar enormemente la eficiencia de esas actualizaciones. En otras palabras, las transacciones permiten hacer scripts más rápido y potencialmente más seguros cuando se usan apropiadamente, ya que nos simplifican la tarea de mantener la consistencia de los datos.

Desafortunadamente, no todas las bases de datos soporta transacciones, por lo PDO debe ejecutarse en lo que se conoce como modo "auto-commit" cuando se abra por primera vez la conexión. Auto-commit significa el modo de que cada consulta que se ejecuta tiene su propia transacción implícita, si la base de datos es compatible con él, o de la transacción si la base de datos no

es compatible con las transacciones. Si necesita una operación, debe utilizar PDO:: beginTransaction() para iniciar una. Si el driver subyacente no es compatible con las transacciones, una excepción de tipo PDOException será lanzado (independientemente de la configuración de control de errores: es siempre una condición de error grave). Una vez que esté en una transacción, debe utilizar PDO:: commit() o PDO::rollBack() para terminar, dependiendo del éxito del código que se ejecuta durante la transacción.

Cuando termina la secuencia de comandos o cuando una conexión está a punto de ser cerrada, si hay una transacción pendiente, PDO automáticamente deshará los cambios hechos hasta el momento. Esta es una medida de seguridad para ayudar a evitar incoherencias en los casos en que el script termine de forma inesperada - si no se confirma explícitamente la transacción, entonces se asume que algo salió mal, por lo que la vuelta atrás se realiza por la seguridad de los datos.

```
<?php
try {
  $dbh = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2',
      array(PDO::ATTR_PERSISTENT => true));
  echo "Connected\n";
} catch (Exception $e) {
  die("Unable to connect: " . $e->getMessage());
}

try {
  $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $dbh->beginTransaction();
  $dbh->exec("insert into staff (id, first, last) values (23, 'Joe',
'Bloggs')");
  $dbh->exec("insert into salarychange (id, amount, changedate)
      values (23, 50000, NOW())");
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

No hay ninguna limitación al tipo de query a realizar en una transacción, también se pueden realizar consultas complejas para extraer los datos, y, posiblemente, utilizar esa información para construir más actualizaciones y consultas, mientras que la transacción está activa, se le garantiza que nadie más puede hacer cambios mientras se está en el medio del trabajo. Para leer más sobre las operaciones, consulte la documentación proporcionada por el

servidor de base de datos.

## Sentencias preparadas y procedimientos almacenados

Muchas de las bases de datos más maduras soportan el concepto de declaraciones preparadas. ¿Qué son? Se puede considerar como una especie de plantilla de elaborarse para el SQL que una aplicación quiere correr, que se pueden personalizar con parámetros variables. Declaraciones preparadas ofrecen dos ventajas principales:

* La consulta sólo tiene que ser analizado (o preparada) una vez, pero puede ser ejecutado varias veces con los mismos parámetros o diferentes. Cuando la consulta se prepara, la base de datos a analizar, compilar y optimizar el plan para ejecutar la consulta. Para consultas complejas de este proceso puede tomar bastante tiempo hasta que perceptiblemente se ralentice una aplicación si es necesario repetir la misma consulta muchas veces con diferentes parámetros. Mediante el uso de una declaración preparada la aplicación evita repetir el ciclo de análisis / compilar / optimizar. Esto significa que las declaraciones preparadas utilizar menos recursos y por lo tanto corren más rápido.

* Los parámetros para comandos preparados no son necesarios que se encomillen, el controlador se encarga de automatizar esto. Si una aplicación utiliza exclusivamente declaraciones preparadas, el desarrollador puede estar seguro de que no se producirá de inyección de SQL (sin embargo, si otras partes de la consulta se está construyendo con la entrada sin escapar, la inyección SQL es posible).

Las declaraciones preparadas son tan útiles que son la única característica que PDO emula a los drivers que no los soportan. Esto asegura que una aplicación será capaz de utilizar el paradigma de los mismos datos de acceso, independientemente de las capacidades de la base de datos.

El siguiente ejemplo usa una consulta preparada con parámetros por nombre:

```php
<?php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (:name,
:value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insert one row
$name = 'one';
$value = 1;
$stmt->execute();
```

```
// insert another row with different values
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

El siguiente ejemplo usa una consulta preparada con parámetros por posición:

```
<?php
$stmt=$dbh->prepare("INSERTINTOREGISTRY(name,value)VALUES(?,?)");
$stmt->bindParam(1,$name);
$stmt->bindParam(2,$value);

//insertonerow
$name='one';
$value=1;
$stmt->execute();

//insertanotherrowwithdifferentvalues
$name='two';
$value=2;
$stmt->execute();
?>
```

## Errores y manejo de errores

PDO ofrece tres diferentes estrategias de control de errores, para adaptarse a su estilo de desarrollo de aplicaciones.

PDO:: ERRMODE_SILENT

Este es el modo por defecto. PDO simplemente establece el código de error para que inspeccione usando PDO::errorCode() y PDO::errorInfo() tanto en la query para los objetos de base de datos, si el error como resultado de una llamada en un objeto declarado, que se invocan con PDOStatement:: errorCode () o PDOStatement::errorInfo () en ese objeto. Si el error es el resultado de una llamada en el objeto de base de datos, se invocan los métodos en el objeto de base de datos en su lugar.

PDO:: ERRMODE_WARNING

Además de establecer el código de error, PDO emitirá un mensaje E_WARNING tradicional. Esta opción es útil durante la depuración / testing, si lo que desea es ver qué problemas se produjeron sin interrumpir el flujo de la aplicación.

PDO:: ERRMODE_EXCEPTION

Además de establecer el código de error, PDO lanzará un PDOException y establecerá sus propiedades a fin de reflejar el código de error e información de error. Esto también es útil durante la depuración, ya que efectivamente "levanta" la secuencia de comandos en el punto del error, señalando a los posibles problemas en el código (recuerde: las transacciones se revierten automáticamente cuando se genera una excepción que no es atrapada).

El modo de excepción también es útil porque se puede estructurar el manejo de errores con más claridad que con los tradicionales avisos de PHP, y con menos código anidado y explícitamente comprobar el valor de retorno de cada llamada base de datos.

Si estamos implementando un sistema Orientado a Objetos, este es el modo que debemos usar para no salirnos del paradigma.

```
<?php
  $dbh = new PDO( /* your connection string */ );
  $dbh->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING );
  // . . .
?>
```

## La clase PDO

Representa una conexión entre PHP y un servidor de base de datos.

La Clase tiene la siguiente estructura:

```
PDO {
        __construct ( string $dsn [, string $username [, string $password [, array $driver_options ]]] )
        bool beginTransaction ( void )
        bool commit ( void )
        mixed errorCode ( void )
        array errorInfo ( void )
        int exec ( string $statement )
        mixed getAttribute ( int $attribute )
        array getAvailableDrivers ( void )
        bool inTransaction ( void )
        string lastInsertId ([ string $name = NULL ] )
        PDOStatement prepare ( string $statement [, array $driver_options = array() ] )
        PDOStatement query ( string $statement )
        string quote ( string $string [, int $parameter_type = PDO::PARAM_STR ] )
        bool rollBack ( void )
        bool setAttribute ( int $attribute , mixed $value )
```

```
}
```

La función de cada método se resume a continuación:

* PDO::beginTransaction — Inicia una transacción
* PDO::commit — Completa una transacción
* PDO::__construct — Crea una instancia de PDO representando una conección a la base de datos.
* PDO::errorCode — Trae el SQLSTATE asociado con la última operación sobre el manejador de la base de datos.
* PDO::errorInfo — Trae la información extendida asociada con la última operación sobre el manejador de la base de datos.
* PDO::exec — Ejecuta una sentencia SQL y devuelve el número de filas afectadas.
* PDO::getAttribute — Devuelve un atributo de la conexión.
* PDO::getAvailableDrivers — Devuelve un vector con los drivers PDO disponibles.
* PDO::inTransaction — Verifica si se está en el contexto de una transacción.
* PDO::lastInsertId — Devuelve el ID de la última fila insertada o de una secuencia de valores.
* PDO::prepare — Prepara una sentencia para ejecución y devuelve un Objeto de tipo *sentencia PDOStatement*.
* PDO::query — Ejecuta una sentencia SQL, devolviendo un conjunto resultado de Objetos PDOStatement.
* PDO::quote — Encomilla un string para usando en una query/
* PDO::rollBack — Deshace una transacción.
* PDO::setAttribute — asigna un atributo.

## La clase PDOStatement

Representa una sentencia preparada y, después de la instrucción se ejecuta, en un resultado asociado.

La Clase tiene la siguiente estructura:

```
PDOStatement   implements Traversable   {
  /* Propiedades */
  readonlystring $queryString;
  /* Métodos */
  bool bindColumn ( mixed $column , mixed &$param [, int $type [, int $maxlen [,
mixed $driverdata ]]] )
  bool bindParam ( mixed $parameter , mixed &$variable [, int $data_type =
PDO::PARAM_STR [, int $length [, mixed $driver_options ]]] )
  bool bindValue ( mixed $parameter , mixed $value [, int $data_type =
PDO::PARAM_STR ] )
```

```
  bool closeCursor ( void )
  int columnCount ( void )
  bool debugDumpParams ( void )
  string errorCode ( void )
  array errorInfo ( void )
  bool execute ([ array $input_parameters ] )
  mixed fetch ([ int $fetch_style = PDO::FETCH_BOTH [, int $cursor_orientation =
PDO::FETCH_ORI_NEXT [, int $cursor_offset = 0 ]]] )
  array fetchAll ([ int $fetch_style = PDO::FETCH_BOTH [, mixed $fetch_argument [,
array $ctor_args = array() ]]] )
  string fetchColumn ([ int $column_number = 0 ] )
  mixed fetchObject ([ string $class_name = "stdClass" [, array $ctor_args ]] )
  mixed getAttribute ( int $attribute )
  array getColumnMeta ( int $column )
  bool nextRowset ( void )
  int rowCount ( void )
  bool setAttribute ( int $attribute , mixed $value )
  bool setFetchMode ( int $mode )
}
```

Propiedades:

queryString: Se utiliza una cadena de consulta.

Un resumen del comportamiento de cada método es el siguiente:

* PDOStatement->bindColumn — Liga una columna a una variable PHP
* PDOStatement->bindParam — Liga un parámetro a el nombre de la variable dada.
* PDOStatement->bindValue — Liga un valor a un parámetro.
* PDOStatement->closeCursor — Cierra el cursos, habilitando que la sentencia pueda ser ejecutada de nuevo.
* PDOStatement->columnCount — Devuelve el número de columnas en el conjunto resultado.
* PDOStatement->debugDumpParams — Vuelca un comando SQL preparado.
* PDOStatement->errorCode — Trae el SQLSTATE asociado con la última operación sobre el handler dado.
* PDOStatement->errorInfo — Trae la información de error extendida de la última operación sobre el handler dado.
* PDOStatement->execute — Ejecuta una sentencia preparada.
* PDOStatement->fetch — Trae la siguiente fila desde el conjunto resultado.
* PDOStatement->fetchAll — Devuelve un vector conteniendo todas las filas del conjunto resultado.
* PDOStatement->fetchColumn — Devuelve una columna de la siguiente fila del conjunto resultado.
* PDOStatement->fetchObject — Trae la siguiente fila y la devuelve como un Objeto.

* PDOStatement->getAttribute — Obtiene un atributo de la sentencia.
* PDOStatement->getColumnMeta — Devuelve los metadatos para una columa del conjunto resultado.
* PDOStatement->nextRowset — Avanza a la siguiente fila en un handler que tenga varias filas.
* PDOStatement->rowCount — Devuelve el número de filas afectadas por la última sentencia SQL.
* PDOStatement->setAttribute — Asigna un atributo a una sentencia.
* PDOStatement->setFetchMode — Asigna el modo de devolución por defecto para esta sentencia: `FETCH_COLUMN`, `FETCH_CLASS` y `FETCH_INTO`.

**Drivers PDO estandar**

| Nombre controlador | Soporte bases de datos |
|---|---|
| PDO_CUBRID | Cubrid |
| PDO_DBLIB | FreeTDS / Microsoft SQL Server / Sybase |
| PDO_FIREBIRD | Firebird/Interbase 6 |
| PDO_IBM | IBM DB2 |
| PDO_INFORMIX | IBM Informix Dynamic Server |
| PDO_MYSQL | MySQL 3.x/4.x/5.x |
| PDO_OCI | Oracle Call Interface |
| PDO_ODBC | ODBC v3 (IBM DB2, unixODBC and win32 ODBC) |
| PDO_PGSQL | PostgreSQL |
| PDO_SQLITE | SQLite 3 and SQLite 2 |
| PDO_4D | 4D |