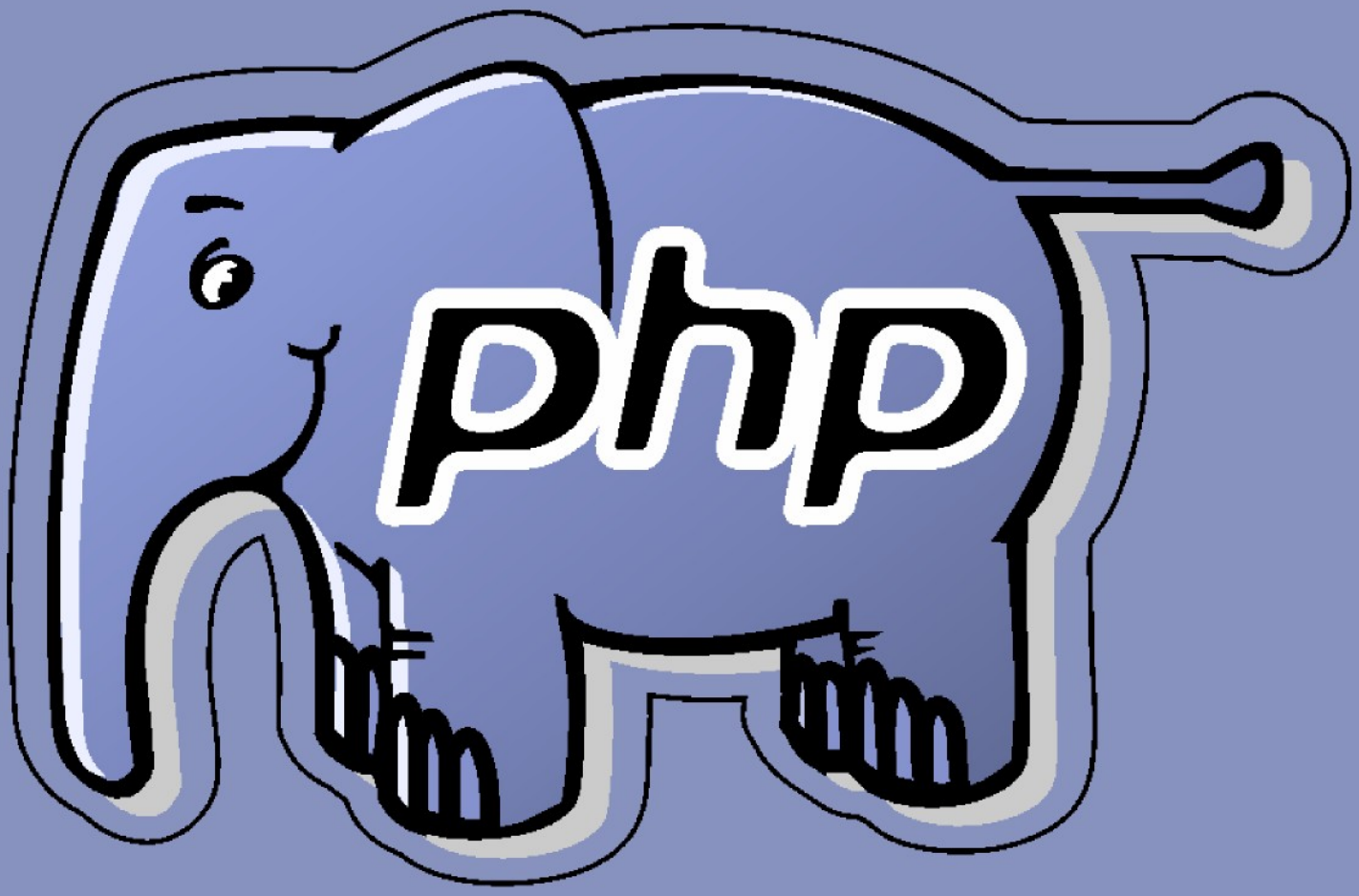
Programación en



Versión 2.0

php

Gugler
Laboratorio de Investigación Gugler

FCyT
Facultad de Ciencia y Tecnología

UADER
Universidad Autónoma de Entre Ríos

CONTENIDO

Capítulo 4

Programación de base de datos en PHP

Introducción

Como habíamos comentado en capítulos anteriores, PHP nace como un lenguaje para la web, es decir, para darle dinamismo a la web. Uno de los significados de esto es el hecho de poder generar contenido dinámico en base a consultas a datos almacenados en bases de datos.

A lo largo de este capítulo estudiaremos las distintas herramientas que nos provee el lenguaje para poder tener acceso a los datos almacenados en bases de datos y nos centraremos posteriormente y en forma particular en el motor de base de datos MySQL y en los conjuntos de funciones, clases y/o métodos que encontraremos en PHP para trabajar con este.

Extensiones de bases de datos

Hasta el momento hemos visto algunas de las funciones que PHP nos ofrece para llevar

a cabo distintas tareas involucradas en el desarrollo web. Este conjunto de funciones aparece de manera predeterminada en el lenguaje ya que se encuentra incluido en el núcleo del mismo.

Existe otro tipo de funciones en forma de *bibliotecas añadidas*, las cuales deben ser instaladas y habilitadas en el sistema de forma específica para poder hacer uso de las mismas. A este tipo de funciones se las denomina *extensiones del lenguaje* y las encontramos como "extensiones" en Window o como "módulos" en GNU/Linux.

Es importante mencionar que el lenguaje posee una gran cantidad de módulos que brindan distintas funcionalidades a PHP. Podemos enumerar los más relevantes como: extensiones criptográficas, manejo de gráficos, correo electrónico, biblioteca estándar, compresión, archivos, tarjeta de crédito, sesiones, entre otros.

También existe la posibilidad de crear extensiones propias para PHP y estas se desarrollan en otro lenguaje denominado C.

De todas las extensiones del lenguaje una de las más importantes y con mayor desarrollo son las extensiones de base de datos que veremos en este capítulo.

Las *Extensiones de base de datos* son el conjunto de librerías que nos permiten ampliar la funcionalidad de lenguaje para poder manipular datos almacenados en distintas bases de datos.

PHP trabaja con dos tipos o categorías de extensiones para base de datos:

- capas de abstracción
- específicas de cada vendedor

Capas de abstracción

Son extensiones que ponen a disposición interfaces para acceder a varias bases de datos. Esto nos permite abstraernos de la base de datos en sí, de manera que si en alguna fase de desarrollo necesitamos cambiar la misma, no se verá afectada la lógica de la aplicación.

Algunas de las capas de abstracción que encontraremos en PHP:

- **DBA:** es una capa general de abstracción para bases de datos basados en archivos. Su funcionalidad se limita a un subconjunto de características comunes entre estas con el apoyo de las bases de datos modernas, como Oracle Berkeley DB.
- **dbx:** es una capa de abstracción de base de datos (donde *X* es alguna de las bases de datos soportadas). Las funciones dbx permiten acceder a todas las bases de datos mediante una convención de llamada única. El funciones dbx no hacen interfaces directamente con las bases de datos, sino que se apoyan en los módulos que sí lo hacen.

- **ODBC:** nos permite acceder varias bases de datos que soportan la API ODBC. En lugar de tener vario drivers similares para distintas bases de datos se agruparon todos bajo ODBC simplificando de esta forma la conexión con las mismas.
- **PDO(PHP Data Objects):** define una interfaz ligera para el acceso a bases de datos en PHP. Proporciona una capa de abstracción de acceso la cual independientemente de la base de datos que está utilizando, invoca las mismas funciones para realizar consultas y obtener datos.

Específicas de cada vendedor

Este tipo de extensiones no crean una capa de abstracción entre nuestra aplicación y la base de datos sino que nos brindan un conjunto de funciones o clases/métodos especificas por cada base de datos que encontramos.

Algunas de las bases de datos para las que se ofrece extensiones:

- dBase
- DB++
- Firebird/InterBase
- Informix
- Mssql — Microsoft SQL Server
- MySQL
- OCI8 — Oracle OCI8
- Paradox — Paradox File Access
- PostgreSQL
- Sybase

Introducción al motor de base de datos MySQL

Es un gestor de base de datos relacional fácil de usar e increíblemente rápido. Es además uno de los motores de base de datos más usados en Internet, la principal razón de esto es que es gratis para aplicaciones no comerciales. Y que posee una importante cantidad de documentación en diversos idiomas y una gran comunidad que ofrece un excelente soporte.

Integrante de los denominados sistemas LAMP (**L**inux, **A**pache, **M**ySql y **P**HP), es desarrollado en 1995 por la empresa opensource MySQL AB quien se propone como objetivo el desarrollo de una herramienta que cumpla el estándar SQL, pero sin sacrificar velocidad, fiabilidad o usabilidad.

Funcionalidades

- **Es un gestor de base de datos:** una base de datos es un conjunto de datos y un gestor de base de datos es una aplicación capaz de administrar este conjunto de datos.
- **Es una base de datos relacional:** una base de datos relacional es un conjunto de datos que están almacenados y relacionados entre si. Para usar y gestionar una base de datos relacional se usa el lenguaje estándar de programación SQL.
- **Es Software Libre:** el código fuente de se puede descargar y está accesible a cualquiera, por otra parte, usa la licencia GPL para aplicaciones no comerciales.
- **Es una base de datos muy rápida, segura y fácil de usar:** la base de datos se ha ido mejorando optimizándose en velocidad. Por eso es una de las bases de datos más usadas en Internet.
- **Existe una gran cantidad de software que la usa:** entre los programas que podemos mencionar encontramos Wordpress, Drupal, Moodle, Joomla, entre otros.

Características

Algunas de las características que nos ofrece MySQL:

- Multiplataforma.
- Posee diversos conectores.
- Incorpora herramienta de modelado, administración y monitoreo.
- Soporta varios formatos de tablas.

- Replicación.
- Particionamiento.
- Procedimientos almacenados.
- Vistas.
- Seguridad.
- Posee estabilidad.
- De simple administración.
- Soporte técnico.
- Licencia Libre o comercial.
- Manuales y ayuda en varios idiomas.
- Posee una gran comunidad.

Tipos de Datos en MySQL

MySQL admite distintas clases de datos, las cuales pueden ser agrupadas según su tipo:

1. **Cadena de Caracteres**

    - CHAR almacena una cadena de longitud fija. La cadena podrá contener desde 0 a 255 caracteres.

    - VARCHAR: almacena una cadena de longitud variable. La cadena podrá contener desde 0 a 255 caracteres. .

    - BLOB-TEXT: se utilizan para almacenar datos binarios como pueden ser ficheros.

    - ENUM: campo que puede tener un único valor de una lista que se especifica. El tipo Enum acepta hasta 65535 valores distintos.

    - SET: un campo que puede contener ninguno, uno o varios valores de una lista. La lista puede tener un máximo de 64 valores.

    - TINYTEXT Y TINYBLOB: Columna con una longitud máxima de 255 caracteres.

    - MEDIUMBLOB Y MEDIUMTEXT: un texto con un máximo de 16.777.215 caracteres.

2. **Numéricos**

    - TINYINT: es un número entero con o sin signo. Con signo el rango de valores

válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255

- BIT Ó BOOL: un número entero que puede ser 0 ó 1

- SMALLINT: número entero con o sin signo. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.

- MEDIUMINT: número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.

- INTEGER, INT: número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295

- BIGINT: número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.

- FLOAT: número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.

- XREAL, DOUBLE: número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308

- DECIMAL, DEC, NUMERIC: Número en coma flotante desempaquetado. El número se almacena como una cadena

3. Fecha y Hora

- DATE: tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia

- DATETIME: Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos

- TIMESTAMP: Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037.

- TIME: almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'

- YEAR: almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

Tipos de Tablas en MySQL

MySQL admite varios motores de almacenamiento los cuales tratan en distintas tablas los datos a manipular. Una de las principales diferencia que encontraremos es que algunos hacen uso de tablas transaccionales y otros no.

A continuación veremos los distintos tipos de tablas que encontraremos en MySQL:

- ISAM: es el formato de almacenaje mas antiguo y en desuso en la actualidad. Presentaba limitaciones (los ficheros no eran transportables entre máquinas con distinta arquitectura, no podía manejar ficheros de tablas superiores a 4 gigas).

- MyISAM: vino a suplantar a ISAM. Proporciona almacenamiento y recuperación de datos rápida ya que hace uso de *tablas no transaccionales.* Es el motor de almacenamiento por defecto a no ser que se tenga una configuración distinta a la que viene por defecto con MySQL.

- MERGE: este motor es conocido también como MRG_MyISAM, es una colección de tablas MyISAM idénticas que pueden usarse como una. "Idéntica" significa que todas las tablas tienen información de columna e índice idéntica. De esta manera podremos definir distintas tablas con exactamente la misma estructura y luego declarar una tabla tipo *MERGE* que las referencie a todas. Luego, cuando queramos hacer una lectura o insertar un dato en todas las tablas, lo hacemos a través de la tabla *MERGE* declarada.

- HEAP – MEMORY: crea tablas con contenidos que se almacenan en memoria. Cada tabla MEMORY está asociada con un fichero de disco. Como indica su nombre, las tablas MEMORY se almacenan en memoria y usan índices hash por defecto. Esto las hace muy rápidas, y muy útiles para crear tablas temporales. Sin embargo, cuando se apaga el servidor, todos los datos almacenados en las tablas MEMORY se pierde.

- InnoDB: dota a MySQL de un motor de almacenamiento transaccional con capacidades de commit (confirmación), rollback (cancelación) y recuperación de fallas. Realiza bloqueos a nivel de fila y también proporciona funciones de lectura consistente sin bloqueo al estilo Oracle en sentencias SELECT. InnoDB también soporta restricciones FOREIGN KEY.

- BerkeleyDB: este motor de almacenamiento proporciona tablas transaccionales. La forma de usar estas tablas depende del modo autocommit:

    - si está ejecutando con autocommit activado (por defecto), los cambios en las tablas se efectúan inmediatamente y no pueden deshacerse.

    - si está ejecutando con autocommit desactivado, los cambios no son permanentes hasta que ejecuta un comando COMMIT . En lugar de hacer un commit puede ejecutar ROLLBACK para olvidar los cambios.

SQL – Lenguaje de consulta estructurado

Introducción

SQL es el lenguaje de manipulación más común utilizado en bases de datos relacionales. Desarrollado entre 1974 y 1975 en IBM Research y llamado por esos años como SEQUEL (Structured English QUEry Language), se convierte en un estándar ANSI en al año 1986 y en 1987 en un estándar ISO. Es introducido por primera vez en una base de datos comercial en el año 1979 por Oracle .

SQL se utiliza para definir, consultar y actualizar la base de datos, y es el más popular de su estilo. Conceptualmente, SQL es un lenguaje de definición de datos (LDD), un lenguaje de definiciones de vistas (LDV) y un lenguaje de manipulación de datos (LMD), que posee también capacidad para especificar restricciones y evolución de esquemas.

A lo largo de este apartado veremos una serie de instrucciones básicas SQL y su sintaxis. Cabe aclarar que este curso no pretende profundizar sobre este lenguaje por lo que se tratarán temas necesario para el desarrollo de las siguientes unidades.

Creando bases de datos y tablas

Contamos con dos instrucciones para realizar estas tareas. En primer lugar, si quisiésemos crear una nueva base de datos:

CREATE DATABASE <nombre>;

Ejemplo:

CREATE DATABASE personal;

Como se puede ver, es una instrucción relativamente sencilla. Ahora, la creación de tablas ya implica instrucciones un poco más complejas. La sintaxis básica es la siguiente:

CREATE TABLE <nombre> (
<columna1> <tipo> [<atributos>],
<columna..n> <tipo> [<atributos>]
)

Ejemplo:

```
CREATE TABLE personas (
    idpersona INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

```
    documento INT,
    apellido VARCHAR(255),
    nombres VARCHAR(255),
    direccion VARCHAR(255)
)
```

Creando índices dentro de las tablas

Al crear una columna con el atributo PRIMARY KEY se genera automáticamente un índice por ese campo para poder optimizar las consultas que se realicen sobre esa tabla. Puede encontrarse con la necesidad de agregar nuevos índices dentro de una tabla para realizar consultas sobre esos campos y así obtener mejores tiempos de respuesta. Haremos uso de la instrucción *CREATE INDEX* para esto. A modo de ejemplo crearemos un indice sobre una columna de un tabla del ejemplo anterior.

```
CREATE INDEX idx1_documento ON personas (documento);
```

Borrando Bases de datos y Tablas

Borrar un objeto de un esquema es sencillo. Haremos uso de la instrucción *DROP* para esto. A modo de ejemplo borraremos la tabla y la base de datos del ejemplo anterior.

```
DROP TABLE personas;
DROP SCHEMA personal;
```

Agregar y manipular datos

Una vez que definimos nuestra base de datos y nuestras tablas nos encontraremos con la necesidad de insertar datos dentro de estas últimas. Para realizar esta tarea contamos con la instrucción INSERT. A continuación veremos dos formas que encontraremos para insertar datos.

INSERT INTO <nombreTabla> VALUES (<valor1>, ..., <valor..n>)

INSERT INTO <*nombreTabla*>(<columna1>, ..., <columna..m>])
VALUES (<valor1>, ..., <valor..n>])

Cuando queramos insertar absolutamente todos los valores correspondientes a cada uno de los campo que posea la tabla, podremos utilizar la primer sentencia. En este caso deberemos insertar cada uno de los valores en el mismo orden en que aparecen declarados en la tabla. Ejemplo:

Creamos la tabla:

```
CREATE TABLE personas (
        idpersona INT NOT NULL PRIMARY KEY,
        documento INT,
        apellido VARCHAR(255),
        nombres VARCHAR(255),
        direccion VARCHAR(255)
)
```

e insertamos un registro:

```
INSERT INTO personas VALUES (22555333,'Gómez','Francisco','Rivadavia');
```

Ahora si quisiésemos insertar valores en algunos campos, lo haríamos con la segunda instrucción.
Ejemplo

Creamos la tabla:

```
CREATE TABLE personas (
        idpersona INT NOT NULL PRIMARY KEY,
        documento INT,
        apellido VARCHAR(255),
        nombres VARCHAR(255),
        direccion VARCHAR(255)
)
```

e insertamos un registro:

```
INSERT INTO personas (documento,apellido)
VALUES (22555333,'Gómez');
```

Ahora bien, si quisiésemos modificar alguno de los valores podremos hacer uso de la instrucción UPDATE. La sintaxis básica es la siguiente:

UPDATE <nombreTabla> SET columna= <nuevoValor>;

Ejemplo:

UPDATE personas SET nombres= 'antonio';

Esta instrucción actualizará los valores de la columna *nombres* por antonio. El problema es que lo realizará en todos los registros existentes. Si quisiésemos restringir esta instrucción:

UPDATE personas SET nombres= 'antonio' WHERE documento=22555333;

Borrar datos

Para borrar datos contamos con la instrucción DELETE la cual nos permitirá eliminar de la tabla que le especifiquemos un dato o todos los datos almacenados. La sintaxis básica es la siguiente:

DELETE FROM <nombreTalba>;

Ejemplo:

DELETE FROM personas;

Esta instrucción borrará todos los registros de la tabla persona. Nuevamente podremos filtrar esta instrucción haciendo uso de la clausula WHERE.

Ejemplo:

DELETE FROM personas WHERE documento=*22555333*;

Realizando consultas

Esta será sin dudas una de las operaciones que más realizaremos en nuestras aplicaciones. Para consultar datos de una o mas tablas haremos uso de la instrucción SELECT. La sintaxis básica es la siguiente:

*SELECT * FROM <nombreTabla>*

Ejemplo:

SELECT * FROM personas;

Esta instrucción devolverá todos los registros existentes en la tabla *personas.* Además en dicha consulta obtendremos todas las columnas que estén definidas en la tabla.

Si quisiésemos recuperar algunas de las columnas pero no todas, deberemos indicarlas en lugar del *.

Ejemplo:

SELECT documento, apellido FROM personas;

Y si quisiésemos filtrar la búsqueda para no obtener todos los registros, utilizaremos nuevamente la sentencia WHERE.

Ejemplo:

SELECT documento, apellido FROM personas WHERE documento=*22555333*;

Introducción a la capa de abstracción PHP Data Object

Introducción

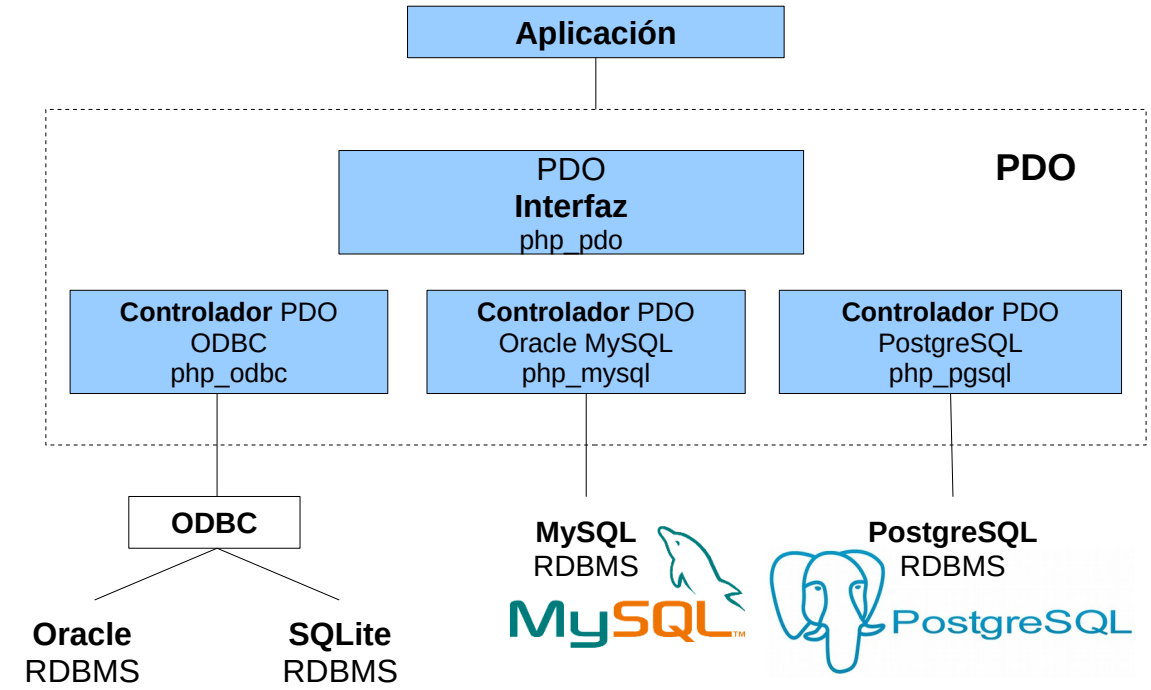
Después de estudiar los distintos tipos extensiones de base de datos, el motor MySQL y el lenguaje de consulta estructurado podemos abordar el tema fundamental del capitulo que se denomina PHP Data Object.

PDO es una extensión de base de datos categorizada como capa de abstracción y que aparece en versiones de PHP5 o superiores. La extensión aparece en el lenguaje en el año 2003 en la primera aparición de PHP5. Y en la versión 5.1 se incorpora como un módulo por defecto en el lenguaje.

Arquitectura

PDO está compuesta por una interfaz principal que dialoga con la aplicación y con los controladores de base de datos. Y los controladores interactúan con los motores de base de datos. Ocasionalmente, podemos encontrar una particularidad en la arquitectura donde vemos que la interfaz PDO se comunica con otra capa de abstracción como ODBC. De este último caso podemos decir que tenemos una doble capa de abstracción sobre la base de datos.

Para detallar la arquitectura mostramos un gráfico de la misma:



Dibujo 1: Arquitectura PDO

Controladores de base de datos PDO

La capa de abstracción PDO necesita de controladores de base de datos para interactuar con ellas y actualmente PDO posee los siguientes:

- PDO_DBLIB para base de datos de tipo FreeTDS / Microsoft SQL Server / Sybase
- PDO_FIREBIRD para motores de tipo Firebird e Interbase 6.x
- PDO_IBM para gestores de base de datos de tipo IBM DB2
- PDO_INFORMIX para motores del Informix Dynamic Server
- PDO_MYSQL para base de datos del Oracle MySQL 3.x, 4.x y 5.x
- PDO_OCI para gestores del tipo Oracle Call Interface
- PDO_ODBC para conectarse con tecnlogías ODBC v3
- PDO_PGSQL para motores de base de datos PostgreSQL
- PDO_SQLITE para gestores SQLITE versión 2 y 3
- PDO_4D para base de datos 4D

Clases y métodos de la capa de abstracción PDO

PDO se compone de tres clases para administrar el acceso a las base de datos de los distintos motores que mencionamos anteriormente.

Las clases son: PDO, PDOStatement y PDOException. La clase PDO representa la conexión a la base de datos y permite manejar la conexión. La clase PDOStatement permite representar una declaración SQL preparada, ejecutarla y luego manipularla por medio de un conjunto de resultados (resultset). La clase PDOException brinda la posibilidad de capturar errores excepcionales en las clases de la extensión PDO.

Cada una de estas clases definen un conjunto de métodos que permiten modificar el comportamiento de la conexión, las declaraciones SQL, los conjuntos de resultados y las excepciones.

A continuación se presentaran las funcionalidad de los métodos de la clase PDO de la capa de abstracción. Todos los métodos de la clase PDO responden a operaciones sobre las conexiones a base de datos. A continuación un detalle de cada uno de los métodos:

- PDO::**beginTransaction**() permite dar inicio a una transacción sobre una conexión de base de datos.
- PDO::**commit**() hace efectiva la transacción en la conexión de la base de datos.
- PDO::**__construct**() permite crear instancias de la clase PDO, cada instancia representa una conexión a una base de datos.

- PDO::**errorCode**() devuelve el código SQLSTATE de la última operación que se llevo a cabo en la conexión del motor de base de datos.
- PDO::**errorInfo**() devuelve información ampliada del error ocurrido en la última operación sobre la conexión al motor de la DB.
- PDO::**exec**() ejecuta una instrucción SQL y devuelve la cantidad de filas afectadas con la sentancia SQL.
- PDO::**getAttribute**() devuelve el valor de una atributo de la conexión al motor de base de datos.
- PDO::**getAvailableDrivers**() retorna los controladores que actualmente están disponibles en la capa de abstracción PDO.
- PDO::**lastInsertId**() devuelve el identificador de la última fila insertada o secuencia de valores de la conexión al motor de la BD.
- PDO::**prepare**() prepara una instrucción SQL para luego ser ejecutada con otro método de la clase PDOStatement.
- PDO::**query**() retorna una instancia de PDOStatement con el conjunto de resultados obtenidos de una consulta pasada por parámetro.
- PDO::**quote**() permite entrecomillar cadenas de textos principalmente consultas SQL.
- PDO::**rollBack**() realiza una vuelta hacia atrás sobre una transacción del motor de la base de datos.
- PDO::**setAttribute**() permite establecer una valor sobre los distintos atributos de la conexión a un motor de base de datos.

Instalación y configuración de PDO

Después de comprender el concepto de PDO y sus distintas clases podemos ver como instalar la capa de abstracción en el lenguaje PHP. Por suerte, la extensión PDO viene instalada por defecto en las actuales versiones de PHP.

Para sistemas operativos GNU/Linux y Windows la implementación PDO viene instalada por defecto con el driver de SQLITE. Si necesitamos acceder a otro motor de base de datos debemos configurar el lenguaje PHP que agregar el soporte del gestor de base de datos.

Generalmente, la instalación de un nuevo controlador de base de datos PDO se realiza modificando el archivo de configuración php.ini del lenguaje. Simplemente, se debe descomentar la opción correspondiente al controlador que deseamos activar en la configuración del lenguaje. Finalmente, debemos reiniciar el servidor Web para que tome los cambios de la configuración.

A continuación dejamos como ejemplo la activación del controlador MySQL PDO en el archivo de configuración php.ini:

#extension=php_pdo_mysql.dll (En Windows)

#extension=php_pdo_mysql.so (En GNU/Linux)

Para activar el controlador del ejemplo anterior hay que borrar el signo cardinal (#) de la linea correspondiente.

Para verificar que tenemos el controlador activado por ejecutar la función phpinfo() o llamar el método PDO::getAvailableDrivers().

Constantes de la capa de abstracción PDO

La implementación PDO propone el uso de constantes para configurar la conexión al motor de base de datos y la confección de consultas SQL.

Podemos encontrar abundantes constantes para el manejo de la configuración de la conexión a los motores de base de datos. También podemos encontrar constantes que permiten configurar las consultas SQL y los resultados de las mismas.

Vamos a mencionar algunas de las constantes que se utilizan para configurar la conexión al motor de base de datos: PDO::ATTR_AUTOCOMMIT (integer), PDO::ATTR_PREFETCH (integer), PDO::ATTR_TIMEOUT (integer), PDO::ATTR_ERRMODE (integer), PDO::ATTR_SERVER_VERSION (integer), PDO::ATTR_CLIENT_VERSION (integer), PDO::ATTR_SERVER_INFO (integer), PDO::ATTR_CONNECTION_STATUS (integer), PDO::ATTR_CASE (integer), PDO::ATTR_CURSOR_NAME (integer), PDO::ATTR_CURSOR (integer), PDO::ATTR_DRIVER_NAME (string), PDO::ATTR_ORACLE_NULLS (integer), PDO::ATTR_PERSISTENT (integer), PDO::ATTR_STATEMENT_CLASS (integer), PDO::ATTR_FETCH_CATALOG_NAMES (integer), PDO::ATTR_FETCH_TABLE_NAMES (integer), PDO::ATTR_STRINGIFY_FETCHES (integer), PDO::ATTR_MAX_COLUMN_LEN (integer), PDO::ATTR_DEFAULT_FETCH_MODE (integer), PDO::ATTR_EMULATE_PREPARES (integer).

Podemos encontrar un listado de todas las constantes de la implementación PDO en el sitio oficial de PHP.

Usando métodos de la clase PDO

Vimos que existen números comportamientos de la clase PDO para manejar las conexiones a los distintos motores de base de datos. En este sección traeremos la mayoría de los métodos de la clase PDO para poder conectarnos a un motor, obtener atributos de la conexión, establecer nuevos valores, hacer consultas SQL sobre el motor y manejar errores que pudieran aparecer.

Conectar PHP con un motor de base de datos utilizando PDO

El constructor de la clase PDO permite crear instancias de la clase y al mismo tiempo crea una conexión con el motor de base de datos especificado. También puede ocurrir que la conexión se realice directamente sobre una base de datos que se encuentra en el motor.

Entonces para poder conectar el lenguaje PHP a una base de datos debemos crear una instancia de la clase PDO utilizando la siguiente sintaxis:

PDO::__construct() (string $dsn [, string $nombreUsuario [, string $contrasenia [, array $opciones_control]]])

El primer parámetro del constructor es obligatorio y refiere al dsn (data source name) que es la fuente de datos. Es una cadena donde se define el motor de base de datos que vamos a utilizar, la base de datos, el equipo y el puerto que utiliza el motor para atender las peticiones. A continuación dejamos un ejemplo de conexión a una base de datos MySQL:

mysql:dbname=**phpn1_db_clase7**;host=**127.0.0.1**;port=**3306**

En lineas generales el formato de la cadena del dsn es el siguiente:

{dbtype}:dbname=**{dbname}**;host=**{dbhost}**;port=**{dbport}**

El segundo parámetro del constructor refiere al nombre del usuario de la base de datos a la cual nos queremos conectar y es opcional ya que algunos motores no indican el dsn.

El tercer parámetro permite ingresar la contraseña del usuario de la base de datos. También es opcional.

El último parámetro del constructor es un array que permite definir opciones de control sobre la conexión de la base de datos. En estas opciones generalmente se establecen atributos a la conexión y se utilizan las distintas constantes de clase que presentamos anteriormente.

Si la llamada al constructor devuelve un objeto PDO la conexión tuvo éxito y caso contrario se devuelve un error y/o una excepción.

Para terminar de comprender como nos conectamos a un motor de base de datos MySQL con la extensión PDO desde PHP vemos el siguiente ejemplo de conexión:

```
<?php

$dsn = 'mysql:dbname=phpn1_db_clase7;host=127.0.0.1;port=3306';
```

```
$user = 'root';

$password = '';

$dbh = new PDO($dsn, $user, $password);

?>
```

Ejecutar sentencias SQL con el método PDO::exec()

Desde la implementación PDO podemos realizar distintas consultas al motor de la base de datos desde varios métodos. Nuestro primer acercamiento a la ejecución de consultas lo veremos con el método PDO::exec() que permite ejecutar directamente instrucciones SQL, el resultado de la consulta devuelve la cantidad de filas afectadas.

El método PDO::exec() responde mejor a las sentencias SQL del tipo INSERT, UPDATE o DELETE que permiten devolver la cantidad de filas afectadas por cualquiera de esas instrucciones. De todas maneras, PDO::exec() puede ejecutar cualquier consulta SQL que se le pase como parámetro.

La sintaxis del método es la siguiente:

```
int PDO::exec  ( string $SqlDeclarada  )
```

Para comprender mejor el uso del método veremos un ejemplo a continuación:

```
<?php

$dsn = 'mysql:host=127.0.0.1;port=3306';

$user = 'root';

$password = '';


$dbh = new PDO($dsn, $user, $password);


$result = $dbh->exec('CREATE DATABASE phpn1_db_clase7;');


$result = $dbh->exec('USE phpn1_db_clase7;');

?>
```

Obtener y establecer valores a los atributos de una conexión a base de datos con PDO

Una vez que tenemos la conexión al motor de base de datos podemos obtener y/o modificar los atributos de la misma, para cualquiera de las operaciones debemos llamar a los métodos PDO::getAttribute() o PDO::setAttribute().

Para los dos casos se recomienda utilizar las constantes de la clase PDO.

A continuación se muestran las sintaxis de ambos métodos:

```
mixed PDO::getAttribute ( int $atributo )

bool PDO::setAttribute ( int $atributo, mixed $valor )
```

Como se puede ver el método PDO::getAttribute() espera como parámetro un valor entero que representa un atributo de la conexión y es aquí donde se sugiere el uso de constantes; el retorno del método puede ser una cadena, un entero u otro tipo de dato (mixed).

El método PDO::setAttribute() espera como parámetro un entero que representa el atributo a modificar y otro entero con el nuevo valor del atributo. Nuevamente en el primer parámetro se recomienda el uso de constantes de la clase PDO. Si se pudo establecer el valor con éxito el método retorno true o false en caso de error.

Para terminar con la explicación de como utilizar los métodos se presenta un ejemplo:

```
<?php

$dsn = 'mysql:host=127.0.0.1;port=3306';


$dbh = new PDO($dsn, 'root', '');

// Muestra información del estado del servidor de base de datos

echo $dbh->getAtributtes(PDO::ATTR_SERVER_INFO).“<br/>”;

// Muestra la versión del gestor de la base de datos

echo $dbh->getAtributtes(PDO::ATTR_SERVER_VERSION).“<br/>”;

// Desactivar el AUTOCOMMIT

$dbh->setAtributtes(PDO::ATTR_AUTOCOMMIT, 0);

// Cambiar el muestreo de errores

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

?>
```

Métodos para mostrar los errores en la conexiones a las DB con PDO

En la clase PDO existen dos métodos que nos pueden informar sobre cualquier error que pudiera ocurrir en las conexiones a las base de datos. Siempre se informa el error sobre la última llamada a cualquier método de la conexión de la base de datos.

El método PDO::errorCode() nos muestra información del error devolviendo el código SQLSTATE del mismo. Generalmente, el código SQLSTATE posee una longitud de 5 caracteres alfanuméricos según lo establecer el estándar ANSI SQL-92. A continuación la sintaxis del método:

mixed **PDO::errorCode** (void)

Si necesitamos más información del error se propone utilizar el método PDO::errorInfo() que devuelve el error en arreglo que tiene los siguientes elementos:

- **0 SQLSTATE** código de error (ANSI 92)
- **1 Código de error** especifico del controlador de base de datos.
- **2 Mensaje de error** especifico del controlador de la base de datos.

La sintaxis del método con información de error ampliada es la siguiente:

array **PDO::errorInfo** (void)

Ahora veremos un ejemplo de como utilizar los métodos que muestran errores en la conexión a una DB:

```
<?php
$dsn = 'mysql:dbname=phpn1_db_clase7;host=127.0.0.1;port=3306';
$user = 'root';
$password = '';

$dbh = new PDO($dsn, $user, $password);
$stmt = $dbh->exec('SELECT * FROM Tabla_Inexistente');
if (!$stmt) {
        echo “Error al intentar conectar con la base de datos.”;
        echo “<br/>C&oacute;digo: ”. $dbh->errorCode();
        echo “<br/>Mensaje del error: “. $dbh->errorInfo()[2];
}
?>
```

Obtener los controladores de base de datos disponibles en la extensión PDO

La interfaz PDO nos permite saber que controladores tenemos disponibles en la extensión PDO utilizando el método PDO::getAvailableDrivers(). Esta operación de la clase PDO nos devuelve un arreglo con los controladores disponibles.

La sintaxis del método es la siguiente:

public static array **PDO::getAvailableDrivers** (void)

Este método puede ser invocado sin la necesidad de instanciar la clase PDO ya que es un método estático. Vemos un ejemplo de uso:

```
<?php
        print_r(PDO::getAvailableDrivers());
?>
```

Cerrando una conexión a base de datos con PDO

De todas las clases que posee la implementación PDO ninguna provee un método para cerrar la conexión a las bases de datos pero para resolver esa necesidad podemos asignar el valor null a la variable que almacena la instancia PDO. También se puede utilizar la función unset().

A continuación vemos un ejemplo para cerrar la conexión:

```
<?php
$dsn = 'mysql:host=127.0.0.1;port=3306';
$user = 'root';
$password = '';

$dbh = new PDO($dsn, $user, $password);

$dbh = null;
?>
```