Acceso a datos en SGBD

Realizado por A.Garay (dpto.de Informática)

Resumen de contenidos

- Introducción
- Conexión a MySQL
- Ejecución de sentencias
- Sentencias preparadas
- Cierre de conexión y gestión de transacciones
- El patrón POST-REDIRECT-GET (PRG)

Introducción (1/2)

- Al igual que en JAVA tenemos JDBC como API de acceso a los SGBD, en PHP, se dispone de PDO (PHP Data Objects).
- En PHP existen drivers para acceder a la mayoría de SGBD conocidos (Oracle, MySql, etc.)
- En la actual presentación trabajaremos con MySql, accediendo desde PHP vía PDO

Introducción (2/2)

- En lugar de PDO, se podrían utilizar instrucciones nativas, tipo **mysql_...(...)**, pero están desaconsejadas porque son menos portables.
- En su lugar, utilizaremos el estándar PDO, que me permite cambiar más fácilmente el motor de BD en un futuro.

Conexión a MySql (1/2)

- Es necesario que una instancia de MySql esté levantada, y tengamos una BD y un esquema accesibles.
  - Ej. host: localhost, usu=root, pwd=, schema=test
- En Línux es conveniente instalar el paquete **php5-mysql**
- En Windows usualmente bastará con descomentar (si no lo está ya) la siguiente línea en el php.ini
  - **extension=pdo_mysql**

Conexión a MySql (2/2)

En cada caso, habrá que concretar los parámetros **$host, $usu, $pwd** y **$schema**, por los valores adecuados para nuestra Base de Datos. Por defecto conecta con **cdcol** (estándar MySql)

```
function conectarMySql(
        $schema='cdcol', $usu='root',
        $pwd='', $host='localhost') {
    try {
        $dsn = "mysql:host=$host;dbname=$schema";
        $db = new PDO ( $dsn, $usu, $pwd );
        return $db;
    } catch ( PDOException $e ) {
        print ('ERROR de conexión') ;
        die ();
    }
}
```

Ejecución de una sentencia DML

- Para **INSERT, UPDATE o DELETE**
- Es más ineficaz que una sentencia preparada si se prevé que se repita en el futuro
- Se utiliza el método exec del objeto PDO, que devuelve el número de filas afectadas

```
$db = conectarMySql ();
$consulta = <<<SQL
        insert into cds(titel,interpret)
        values ('Mi carro','M.Escobar')"
SQL;
$db->exec($consulta);
```

Ejecución de una SELECT

$resultado es un array asociativo cuyas claves son los nombres de las columnas de la query ejecutada, y cuyo contenido es el valor de esa columna para la fila actualmente en proceso.

```
$db = conectarMySql ();
$consulta = 'select * from cds';
$resultado = $db -> query ( $consulta );
if (! $resultado) {
    print ('ERROR al ejecutar query') ;
} else {
    foreach ( $resultado as $fila ) {
        echo $fila ['titel'], '//', $fila ['interpret'], '<br/>';
    }
}
```

Sentencias preparadas (1/3)

Son scripts compilados de SQL que pueden ser personalizados usando parámetros de variables.

- La consulta sólo necesita ser analizada/preparada una vez
- Puede ser ejecutada muchas veces cambiando o no los parámetros
- Se evita repetir el ciclo de análisis/compilación/optimización cada vez que se ejecuta.
- Usan menos recursos y se ejecutan antes.
- Con sentencias preparadas se evitan inyecciones SQL

```
$db = conectarMySql ();
$consulta = "insert into cds(titel,interpret) values (?,?)";
$resultado = $db->prepare ( $consulta );
$resultado->execute ( ['Borriquito','Peret'] );
```

Sentencias preparadas (2/3)

- Se pueden utilizar pseudo-variables (cuyos identificadores comiencen por el carácter “dos puntos”) en la consulta preparada, en lugar del comodín “interrogación”
- De esta manera no es necesario recordar el orden en el que fueron introducidos los parámetros.
- A la hora de ejecutar, habrá que suministrar un array asociativo cuyas claves sean los nombres de los parámetros incluidos en la consulta, y cuyos valores sean los valores reales que queremos introducir.
- Si no queremos utilizar arrays asociativos podemos llamar a **$resultado -> execute()** sin parámetros, siempre y cuando antes vinculemos cada dato con su valor utilizando:
    - **$resultado -> bindParam(‘:nombre’,$valorAInsertar)**

```
$db = conectarMySql ();
$consulta = "insert into cds(titel,interpret) values (:titulo,:artista)";
$resultado = $db->prepare ( $consulta );
$resultado->execute ([':titulo' => 'Mediterráneo',':artista'=>'Serrat']);
$resultado->execute ([':titulo' => 'Wonderwall',':artista'=>'Oasis']);
```

Sentencias preparadas (3/3)

- Cuando se trata de preparar una SELECT hay que hacer un poco de trabajo extra para recibir el resultado en forma de array
- Usar una sentencia preparada no es siempre la manera más eficiente de ejecutar una sentencia.
- Una sentencia preparada ejecutada una sola vez causa más viajes de ida y vuelta desde el cliente al servidor que una sentencia no preparada.
- Es por esta razón por lo que *SELECT* no se suele ejecutar como una sentencia preparada, excepto si es crucial evitar una inyección de dependencias (p.ej. en login)

```php
$db = conectarMySql ();
$consulta = <<<SQL
        select * from cds
        where jahr > ?
SQL;
$sentencia = $db->prepare ( $consulta );
$sentencia->execute ( [1990] );
$resultado = $sentencia->fetchAll();
foreach ($resultado as $fila){
    echo $fila['titel'],' // ',$fila['jahr'],'<br/>';
}
```

El método lastInsertId()

- Es un **método PDO** de la conexión de la BBDD que me devuelve el último autonumérico utilizado en una inserción.
- Es útil para vincular una fila recién insertada con otras de otras tablas (relaciones N a N, normalmente)

Cierre de conexión y transacciones

- Cierre conexión:
  - **$db = null**
  - o bien, esperar a que finalice el script
- Gestión transacciones (sin autocommit)
  - **$db -> beginTransaction();**
  - ….. sentencias SQL
  - **$db -> commit();**

¿POST o GET?

- En principio da igual si enviamos información al servidor vía GET o vía POST. Aparentemente es sólo un tema estético (ver o no ver los datos en el URI).
- En la práctica intentaremos enviar todos los datos de formularios siempre vía POST, para que no sean visibles, y para que no sea fácil repetir por error, operaciones delicadas (ver patrón PRG)
- Sólo enviaremos vía GET datos que no provoquen escrituras en nuestras BDs (para cambiar el comportamiento de nuestra aplicación)
    - P.ej el parámetro “q” en la página de Google, es decir, parámetros para los que queremos que alguien “a mano” o desde otro programa pueda enviar fácilmente, tan sólo manipulando la QUERY_STRING

El “problema del reenvío POST”

- Los datos que se envían desde un formulario **f.html** vía POST a un script **s1.php**, se almacenan en la caché del navegador.
- Cuando **s1.php** devuelva su código HTML de respuesta en el URI se seguirá viendo (en la barra de navegación) la ruta a **s1.php.**
- Si en ese momento refrescamos (p.ej. pulsando F5) estaremos pidiendo que se “re-ejecute” **s1.php** y se reenviarán los datos enviados anteriormente (aunque nos avisará previamente, mediante un alert)
  - También ocurriría lo mismo si volviéramos (botón back) a **f.html**, y volviéramos a **s1.php** (botón forward).
- Esto podría provocar que se re-ejecutara una operación que sólo querríamos que ocurriera una vez (p.ej, alta de un usuario, una compra, etc.).
- Cuando menos, podríamos confundir al usuario con el “alert” de reenvío de formulario.

Patrón PRG: POST-Redirect-GET (1/2)

- Para evitar la situación anterior, lo que podemos hacer, de forma genérica, es lo siguiente.
  - Realizar la operación (normalmente que afecta a la BD) desde **s1.php**
  - Guardar los datos POST en una SESSION (por si fueran necesarios más tarde, para mostrar un mensaje personalizado, por ejemplo)
  - Enviar un header de redirección a una tercera página **s2.php**
  - El navegador cargará s2.php vía GET.
  - Recuperar en **s2.php** los datos vía SESSION (si se necesitaran) y opcionalmente mostrar un mensaje de confirmación, error, etc. en función de cómo haya ido la operación realizada desde **s1.php**
  - Destruir los datos de SESSION y la SESSION en sí

Patrón POST-Redirect-GET (2/2)

- Cuando se trata de operaciones con Bases de Datos en las que simplemente se devuelve una página de confirmación de la transacción, haremos la operación BD en el primer script (POST) y después redirigiremos vía GET a la página de confirmación
- Ver Wikipedia
- Podemos enviar un “nonce” en un campo hidden adicionalmente, para evitar más reenvíos indeseados (ver)