5. Accés a dades amb PHP PDO

Desenvolupament web en entorn servidor

Vicent Jordà

24/11/2020

Índex de continguts

1 Maneig de PHPMyAdmin

2 Accés a dades mitjançant PHP PDO

2.1 Introducció

PHP permet treballar en base de dades MySQL gràcies a dues llibreries. MySQLi i PDO. Com que MySQLi està dissenyada exclusivament per a MySQL nosaltres aprofundirem en PDO, que permet connectar fins amb 12 sistemes gestors de base de dades diferents.

Les sigles PDO (*PHP Data Objects*) fan referència a una interfície de PHP que ens permet accedir a bases de dades de qualsevol tipus en PHP.

Cada controlador de bases de dades que implemente la interfície PDO pot exposar característiques específiques de la base de dades, com les funcions habituals de l'extensió. Cal observar que no es pot realitzar cap de les funcions de la bases de dades utilitzant l'extensió PDO per si mateixa; s'ha d'utilitzar-ne una de PDO específica de la base de dades per tenir accés a un servidor de base de dades.

PDO proporciona una capa d'abstracció d'accés a dades, el que significa que, independentment de la base de dades que s'estiga utilitzant, s'usen les mateixes funcions per fer consultes i obtenir dades.

Per saber els controladors PDO disponibles en el nostre sistema:

```php
print_r (PDO :: getAvailableDrivers ());
```

La informació oficial sobre PDO es troba en <http://php.net/manual/es/book.pdo.php>

2.2 Classes PDO, PDOStatement i PDOException

La classe PDO proporciona 3 classes per a gestionar l'accés a la base de dades

- `PDO`: S'utilitza per representar la connexió entre PHP i un servidor de bases de dades. <http://php.net/manual/es/class.pdo.php>
- `PDOStatement`: representa una sentència preparada i també ens permet accedir al conjunt de resultats associat. <http://php.net/manual/es/class.pdostatement.php>
- `PDOException`: representa els errors generats PDO. <<http://php.net/manual/es/class.pdoexception.php>>

2.3 Connexió a la base de dades

Per a crear una nova connexió s'utilitza la classe `PDO`:

```
1 public __construct(string $dsn [, string $username [, string $passwd [,
      array $options ]]] )
```

El constructor té els següents paràmetres:

- `dsn`: cadena de connexió
- `username`: nom d'usuari
- `passwd`: contrasenya d'usuari
- `options`: permet afegir altres opcions de la connexió

```
1 $pdo = new PDO("mysql:host=localhost; dbname=test", "dbuser", "1234");
```

En l'exemple anterior `MYSQL` representa una connexió a una base de dades MySQL (podria ser: MSSQL, Sybase, sqlite, etc.) anomenada `test` que es troba al servidor `localhost`. La connexió es realitzarà mitjançant l'usuari `dbuser` i la contrasenya `1234`.

```
1  // Exemple de connexió a diferents tipus de bases de dades.
2  # Connectem a la base de dades
3  $host = 'www.local';
4  $hbname = 'cxbasex';
5  $user = 'cxbasex';
6  $pass = 'xxxxxx';
7
8  try {
9      # MS SQL Server y Sybase amb PDO_DBLIB
10     $pdo = new PDO("MSSQL:host=$host;dbname=$dbname, $user, $pass");
11     $pdo = new PDO("Sybase:host=$host;dbname=$dbname, $user, $pass");
12
13     /* MySQL amb PDO_MYSQL
14      Perqué la connexió a mysql utilitzi les collation UTF-8 afegir
           charset = utf8 a string
15      de la connexió. */
```

```php
    $pdo = new PDO  ("mysql:host=$host;dbname=$dbname;charset=utf8", $user
        , $pass);

    // Perqué generi excepcions a l'hora de reportar errors.
    $pdo->setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    # SQLite Database
    $pdo = new PDO ("sqlite:my/database/path/database.db");
}
catch (PDOException $e) {
    die($e-> getMessage ());
}

// Si tot va bé en $pdo tindrem el objecte que gestionarà la connexió amb
    la base de dades.
```

Tancar la connexió a la base de dades

Es recomana tancar sempre la connexió a la base de dades quan no es vaja a utilitzar més durant el nostre procés.

Cal recordar que els recursos són limitats i quan hi ha pocs usuaris no hi ha cap problema, però si tenim molts usuaris simultanis llavors és quan sorgeixen problemes en haver assolit el nombre màxim de connexions amb el servidor, per exemple.

En tancar la connexió de forma explícita accelerem l'alliberament de recursos perquè estiguin disponibles per a altres usuaris.

```php
// Si vullguerem tancar la connexió amb la base de dades simplement podrí
    em fer al final del fitxer.
$pdo = null;
```

2.4 Execució de consultes

Per a realitzar consultes `PDO` proporciona tres mètodes:

- `PDO::query()` per a consultes de recuperació de dades (SELECT).
- `PDO::exec()` per a consultes d'inserció, modificació i esborrat de dades (INSERT, UPDATE i DELETE)
- `PDO::prepare()` per a consultes preparades. Aquest és el mètode recomanat però això hem dedicat un apartat específic.

Una vegada executada la consulta les dades s'obtenen a través del mètode `PDOStatement::fetch()` o `PDOStatement::fetchAll()`.

- `fetch()`: Obté la següent fila d'un recordset (conjunt de resultats). http://php.net/manual/es/pdostatement.fetch.php

- `fetchAll()`: Retorna un array que conté totes les files del conjunt de resultats (el tipus de dades a retornar es pot indicar com a paràmetre). http://php.net/manual/es/pdostatement.fetchall.php

Estil de búsqueda (*fetch style*)

Abans de cridar al mètode `fetch()` una bona idea és indicar-li com volem que ens torne les dades de la base de dades.

Tindrem les següents opcions en el mètode `fetch()`:

- `PDO::FETCH_ASSOC`: Torna les dades en un array associatiu pel nom de camp de la taula.
- `PDO::FETCH_NUM`: Retorna un array indexat per la posició del camp.
- `PDO::FETCH_BOTH`: Retorna un array associatiu pel nom de camp de la taula i un
- indexat per la posició del camp. És una combinació dels dos estils anteriors. És l'estil per defecte.
- `PDO::FETCH_BOUND`: Assigna els valors retornats a les variables assignades amb el mètode `bindColumn()`.
- `PDO::FETCH_CLASS`: Assigna els valors dels camps a les propietats d'una classe. Si les propietats no existeixen en aquesta classe, les crearà.
- `PDO::FETCH_INTO`: Actualitza una instància existent d'una classe.
- `PDO::FETCH_LAZY`: Combina PDO::FETCH_BOTH / PDO::FETCH_OBJ, creant les variables de l'objecte a mesura que es van fent servir.
- `PDO::FETCH_OBJ`: Retorna un objecte anònim amb els noms de les propietats que es corresponen amb els noms de columnes.

Podem ajustar l'estil de búsqueda de dues formes:

Abans de recuperar registres

```
1 $stmt->setFetchMode(PDO::FETCH_ASSOC);
```

O en el moment de recuperar-los:

```
1 $stmt->fetch(PDO::FETCH_ASSOC);
```

```
2 $stmt->fechtall(PDO::FETCH_ASSOC);
```

Cal saber que no podem usar `PDO::FETCH_CLASS` com a paràmetre de `PDOStatement::fetch()`. Cal controlar l'estil de búsqueda amb `PDOStatement::setFetchMode()`.

FETCH_ASSOC

Per executar la consulta SELECT si no tenim paràmetres en la consulta podrem usar `PDO::query()`

Vegem un exemple de consulta SELECT:

```
1  try {
2      #Per executar la consulta SELECT si no tenim paràmetres en la consulta
           podrem usar -> query ()
3      $stmt = $pdo->query ('SELECT name, addr, city from colleague');
4
5      /* Indiquem en quin format volem obtenir les dades de la taula en
          format d'array associatiu.
6         Si no indiquem res per defecte s'usarà FETCH_BOTH el que ens
             permetrà accedir com a vector
7          associatiu o array numéric. */
8      $stmt->setFetchMode(PDO::FETCH_ASSOC);
9
10     #Llegim les dades del recordset amb el métode->fetch ()
11
12     while ($row=$stmt->fetch()) {
13        echo $row['name']. "<br/>";
14        echo $row['addr']. "<br/>";
15        echo $row['city']. "<br/>";
16     }
17
18      #Per alliberar els recursos utilitzats en la consulta SELECT
19      $stmt = null;
20   } catch (PDOException $err) {
21         // Mostrem un missatge genéric d'error.
22     echo "Error: executant consulta SQL.";
23   }
```

FETCH_OBJ

En aquest estil de búsqueda es crearà un objecte estàndard (`stdClass`) per cada fila que llegim del recordset.

Per exemple:

```
1      try {
2          #Creem la consulta
```

```
3           $stmt = $pdo->query('SELECT name, addr, city from colleague');
4           #Ajustem la manera d'obtenció de dades
5           $stmt->setFetchMode(PDO::FETCH_OBJ);
6
7           #Mostrem els resultats.
8           #Fixeu-vos que es torna un objecte cada vegada que es llegeix una
                fila del recordset.
9
10          while ($row = $stmt->fetch()) {
11              echo $row->name. "<br/>";
12              echo $row->addr. "<br/>";
13              echo $row->city. "<br/>";
14          }
15
16          #Alliberem els recursos utilitzats per $stmt
17          $stmt = null;
18      }
19      catch (PDOException $err)
20      {
21      // Mostrem un missatge genéric d'error.
22      echo "Error: executant consulta SQL.";
23      }
```

FETCH_CLASS

En aquest estil de búsqueda els registres es tornaran en una nova instància de la classe indicada en el segon paràmetre, fent correspondre les columnes del conjunt de resultats amb els noms de les propietats de la classe, i cridant al constructor després, a menys que también es proporcione l'opció `PDO::FETCH_PROPS_LATE`. Si algun nom de camp no existeix com a propietat en la classe es crearà dinàmicament.

```
1  class Persona
2  {
3      private $name;
4
5      public function __construct()
6      {
7          $this->decir();
8      }
9
10     public function decir()
11     {
12         if (isset($this->name)) {
13             echo "Soy {$this->name}.\n";
14         } else {
15             echo "Aún no tengo nombre.\n";
```

```
16          }
17      }
18  }
19
20  $sth = $dbh->query("SELECT * FROM people");
21  $sth->setFetchMode(PDO::FETCH_CLASS, 'Persona');
22  $persona = $sth->fetch();
23  $persona->decir();
24  $sth->setFetchMode(PDO::FETCH_CLASS|PDO::FETCH_PROPS_LATE, 'Persona');
25  $persona = $sth->fetch();
26  $persona->decir();
```

Mostrarà:

```
1  Soy Alice.
2  Soy Alice.
3  Aún no tengo nombre.
4  Soy Bob.
```

2.5 Consultes preparades

Com hem vist en l'apartat anterior les consultes poden acceptar paràmetres. LA MILLOR FORMA DE REALITZAR AQUESTES CONSULTES ÉS MITJANÇANT LES CONSULTES PREPARADES.

Les consultes preparades ens aporten dues avantatges importants:

- Per a sentències que seran executades en múltiples ocasions amb diferents paràmetres optimitza el rendiment de la aplicació.
- Ajuda a prevenir injeccions SQL eliminant la necessitat de posar entre cometes manualment els paràmetres.

Per a construir una sentència preparada cal incloure uns marcadors en la sentència SQL.

Hi ha tres formes de fer-ho:

```
1  # Marcadors anònims
2  $stmt = $pdo->prepare("INSERT INTO colleague (name, addr, city) values (?,
       ?, ?)");
3
4  # Marcadors coneguts
5  $stmt = $pdo->prepare("INSERT INTO colleague (name, addr, city) values (:
      name, :addr, :city)");
6
```

```
7  # Sense marcadors. Aquest mètode està desaconsellat! Prohibit en el nostre
       cas.
8  $stmt = $pdo->prepare("INSERT INTO colleague (name, addr, city) values (
       $name, $addr, $city)");
```

Hauràs d'usar el primer o el segon mètode dels que es mostren anteriorment. El tipus de marcadors que utilitzem afectaran a la forma d'assignar els valors d'eixos marcadors.

Assignació amb marcadors anònims

Per enllaçar els marcadors anònims amb el seu corresponent valor es pot utilitzar `bindParam` o `bindValue`:

> ATENCIÓ:`$pdo->prepare()` usant marcadors anònims `?`, tracta totes les variables com si foren *string*, per la qual cosa farà servir cometes per delimitar els seus valors per defecte.

```
1  # Marcadors anònims
2  $stmt = $pdo->prepare("INSERT INTO colleague (name, addr, city) values (?,
       ?, ?)");
3
4  # Assignem variables a cada marcador, indexats l'1 al 3
5  $stmt->bindParam(1, $name);
6  $stmt->bindParam(2, $addr);
7  $stmt->bindParam(3, $city);
8
9  # Inserim una fila.
10 $name = "Daniel";
11 $addr = "1 Wicked Way";
12 $city = "Arlington Heights";
13 $stmt->execute();
14
15 # Inserim un altra fila con valores diferents.
16 $name = "Steve"
17 $addr = "5 Circle Drive";
18 $city = "Schaumburg";
19 $stmt->execute();
```

Un altra forma d'assignació amb marcadors anònims és mitjançant un array associatiu:

```
1   # Les dades que volem inserir
2   $dades = ['Cathy', '9 Dark and Twisty Road', 'Cardiff'];
3
```

```
4   $stmt = $pdo->prepare("INSERT INTO colleague (name, addr, city) values
        (?, ?, ?)");
5   $stmt->execute($dades);
```

Diferència entre l'ús de bindParam i bindValue

Amb `bindParam` es vincula la variable al paràmetre i en el moment de fer l'*execute* és quan s'assigna realment el valor de la variable a aquest paràmetre.

Amb `bindValue` s'assigna el valor de la variable a aquest paràmetre just en el moment d'executar la instrucció `bindValue`.

Exemple de diferència entre `bindParam` i `bindValue`:

```
1  // Ejemplo con bindParam:
2  $sex = 'hombre';
3  $s = $dbh->prepare('SELECT name FROM estudiantes WHERE sexo = :sexo');
4  $s->bindParam(':sexo', $sex);
5  $sex = 'mujer';
6  $s->execute(); // se ejecutó con el valor WHERE sexo = 'mujer'
7
8  // El mismo ejemplo con bindValue:
9  $sex = 'hombre';
10 $s = $dbh->prepare('SELECT name FROM students WHERE sexo = :sexo');
11 $s->bindValue(':sexo', $sex);
12 $sex = 'mujer';
13 $s->execute(); // se ejecutó con el valor WHERE sexo = 'hombre'
```

Assignació amb marcadors coneguts

Els marcadors coneguts són la forma més recomanable de treballar amb PDO, ja que a l'hora de fer el `bindParam` o el `bindValue` es pot especificar el tipus de dada i la seua longitud.

Format de `bindParam` amb marcadors coneguts:

```
1  bindParam(':marcador', $variableVincular, TIPO DATOS PDO)
```

Exemple d'ús de `bindParam`:

```
1  $stmt->bindParam(':calorias', $misCalorias, PDO::PARAM_INT);
2  $stmt->bindParam(':apellidos', $misApellidos, PDO::PARAM_STR, 35); // 35
       caracteres como máximo.
```

Amb marcadors coneguts quedaria de la següent forma:

```php
$stmt = $pdo->prepare("INSERT INTO colleague (name, addr, city) value (:name, :addr, :city)");

# El primer argument de bindParam és el nom del marcador i el segon la variable
# que contindrà les dades.

 # Els marcadors conocidos siempre comienzan con :
$stmt->bindParam(':name', $name);
$name='Pepito';

$stmt->bindParam(':addr', $addr);
$addr='Duanes, 17';

$stmt->bindParam(':city', $city);
$city = 'Pego';

$stmt->execute();

# També podem fer-ho mitjançant un array associatiu
$dades = ['name' => 'Pepito', 'addr' => 'Duanes, 17', 'city' => 'Pego' );

# Fixeu-vos es passa l'array de dades en execute().
$stmt = $pdo->prepare("INSERT INTO colleague (name, addr, city) value (:name, :addr, :city)");
$stmt->execute($dades);

# La última instrucción se podría poner también así:
$stmt->execute([
    'name' => 'Pepito',
    'addr' => 'Duanes, 17',
    'city' => 'Pego'
]);
```

Una altra característica dels marcadors coneguts és que ens permetran treballar amb objectes directament a la base de dades, assumint que les propietats d'aquest objecte coincideixen amb els noms dels camps de la taula a la base de dades.

Exemple d'ús de marcadors coneguts i objectes:

```php
# Un objeto sencillo
class Person {
    public $name;
    public $addr;
```

```php
    public $city;

    function __construct($n,$a,$c) {
        $this->name = $n;
        $this->addr = $a;
        $this->city = $c;
    }
    # etc ...
}

$cathy = new Person('Pepito','Duanes, 17','Pego');

# Preparación de la consulta
$stmt = $pdo->prepare("INSERT INTO colegas (name, addr, city) value (:name
    , :addr, :city)");

# Inserción del objeto
$stmt->execute((array)$cathy);
```

Exemple d'ús

Per a utilitzar les consultes preparades seguirem sempre aquest esquema:

Figure 1: Consultes preparades

```php
$pdo = new PDO('sqlite:/path/db/users.db');

# 1. Preparem la connexió
$stmt = $pdo->prepare('SELECT name FROM users WHERE id = :id');

$id = 5;

# 2. Vinculem els paràmetres
$stmt->bindParam(':id', $id, PDO::PARAM_INT);
```

```
10
11 # 3. Executem la consulta
12  $stmt->execute();
13
14  # 4. Obtenim els registres
15
16  # Un a un
17  $row = $stmt->fetch();
18
19  # Tots a l'hora
20  $rows = $stmt->fetchAll();
```

> Recorda que el tipus DATE de MySQL es recupera com un *string* en format "YYYY-MM-DD" i a l'hora d'emmagatzemar-lo cal que tinga el mateix format.
> Si fóra DATETIME també caldria afegir l'hora "YY-MM-DD hh:mm:ss".

2.6 Inserció, modificació i eliminació de dades de la BBDD

Inserir noves dades, actualitzar-les o esborrar-les són algunes de les operacions més comunes en una base de dades. Amb PDO podem fer aquestes aquestes operacions en 3 passos.

Figure 2: Prepare Bind Execute

Exemple d'ús:

```
1 # $stmt serà un objecte de tipus PDOStatement (consulta preparada)
2 $stmt = $pdo->prepare("INSERT INTO colleague(name, addr, city) values (:
      name, :addr, :city)");
3
4 # Assignem els valors dels marcadors
5 $name = "Josep"
6 $addr = "Duanes, 17"
```

```php
$city = "Pego"

$stmt->BindValue("name", $name);
$stmt->BindValue("addr", $addr);
$stmt->BindValue("city", $city);

# Executem la consulta amb ->execute() método de l'objecte PDOStatement
# Este método devuelve true o false.
$result = $stmt->execute();
```

Exemples CRUD (Create, Read, Update, Delete): INSERT, UPDATE y DELETE

INSERT

```php
try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
       $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $stmt = $pdo->prepare('INSERT INTO colleague (name, addr, city )
       VALUES(:name,
    :addr, :city)');

    $stmt->execute([
    ':name' => 'Josep',
    ':addr' => 'Duanes, 17',
    ':city' => 'Pego'
    ]);

    # Affected Rows?
    echo $stmt->rowCount(); // 1
} catch(PDOException $e) {
    echo 'Error: ' . $e->getMessage();
}
```

UPDATE

```php
$id = 5;
$name = "Joe the Plumber";

try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
       $username, $password);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

```

```
8       $stmt = $pdo->prepare('UPDATE someTable SET name = :name WHERE id = :
            id');
9       $stmt->execute(array(
10      ':id' => $id,
11      ':name' => $name
12      ));
13
14      echo $stmt->rowCount(); // 1
15  } catch(PDOException $e) {
16      echo 'Error: ' . $e->getMessage();
17  }
```

DELETE

```
1   $id = 5; // From a form or something similar
2
3   try {
4       $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8",
            $username, $password);
5       $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
6
7       $stmt = $pdo->prepare('DELETE FROM someTable WHERE id = :id');
8       $stmt->bindParam(':id', $id); // this time, we'll use the bindParam
            method
9       $stmt->execute();
10
11      echo $stmt->rowCount(); // 1
12  } catch(PDOException $e) {
13      echo 'Error: ' . $e->getMessage();
14  }
```

2.7 Alguns mètodes d'interès

Alguns mètodes interessant i que podem utilitzar per a millorar la gestió de l'accés a dades són:

- `PDO::lastInsertId()`, ens torna la clau primària (`id`) del últim registre inserit en la base de dades.
- `PDOStatement::rowCount()`, ens torna el número de files afectades por una sentència DELETE, INSERT, o UPDATE.

2.8 Gestió d'errors en l'accés

PDO pot utilitzar les excepcions per gestionar els errors, el que significa que qualsevol cosa que fem amb PDO podríem encapsular en un bloc try/catch per gestionar si produeix algun error.

Podem forçar PDO perquè treballi en qualsevol de les tres maneres següents:

- `PDO::ERRMODE_SILENT`. És el mode per defecte. Aquí haurem de revisar els errors usant `errorCode()` i `errorInfo()`.
- `PDO::ERRMODE_WARNING`. Genera errors de revisió de resultats PHP però permetria l'execució normal de l'aplicació.
- `PDO::ERRMODE_EXCEPTION`. Serà la forma més utilitzada en PDO. Dispara una excepció permetent-nos gestionar l'error de forma amigable.

```php
//Activació de la manera de treball de PDO
$pdo-> setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
$pdo-> setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);

// Es recomana activar aquesta opció per gestionar els errors amb PDOException
$pdo-> setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Es recomana activar aquesta opció per gestionar els errors amb PDOException, d'altra forma no apareixerà cap missatge i serà complicat detectar-los.

```php
$pdo-> setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

EXEMPLE D'ÚS:

```php

# Connectem a la base de dades
$host = 'www.local';
$dbname = 'cxbasex';
$user = 'cxbasex';
$pass = 'xxxxxx';

try {
    $pdo=new PDO ("mysql:host=$host;dbname=$dbname;charset=utf8", $user, $pass);
    $pdo->setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}
catch (PDOException $e) {
```

```php
    echo "S'ha produït un error en intentar connectar al servidor MySQL:".
        $e->getMessage();
}

try {
    # Un altre Exemple d'error! DELECT en lloc de SELECT!
    $pdo->exec( 'DELECT name FROM people');
}
catch (PDOException $e) {
    echo "S'ha produït un error en l'execució de la consulta:". $e->
       getMessage();

    /* En aquest cas hem mostrat el missatge d'error i, a més emmagatzemem
       en un fitxer els errors generats. */
    file_put_contents ('PDOErrors.txt', $e->getMessage(), FILE_APPEND);
}
```

Exercici pràctic

Accés i consulta a la base de dades

ENUNCIAT

1. Crea la base de dades `movies`.
2. Importa l'arxiu `movies.sql`.
3. Crea l'usuari `dbuser` amb `1234` de constrasenya.
4. Dona-li els privilegis necessaris perquè sols puga fer SELECT, INSERT, DELETE i UPDATE
5. Modifica el projecte perquè obtinga els partners de la base de dades.
6. Modifica també les pel·lícules perquè les obtinga de la base de dades.

OBTIN LA RESPOSTA COM UN ARRAY D'OBJECTES DE LA CLASSE `Movie` AMB EL TIPUS DE RETORN `FETCH_CLASS`.

Exercici pràctic

Consultes preparades

ENUNCIAT

1. Implementa el filtre de búsqueda en `partners.php` (seguint el vídeo)
2. Modifica `single-page.php` perquè obtinga les dades de la base de dades. Rebrà `id` com a paràmetre pel querystring.
3. Modifica el filtratge i l'ordenació en `movies.php` perquè funcione des de la base de dades (OPCIONAL)

UTILITZA CONSULTES PREPARADES SEMPRE QUE CALGA USAR PARÀMETRES. OBTIN LA RESPOSTA COM UN ARRAY D'OBJECTES DE LA CLASSE `Movie` AMB EL TIPUS DE RETORN `FETCH_CLASS`.

Exercici pràctic

Inserint noves pel·lícules

ENUNCIAT {:.no_toc .nocount}

- Seguint les indicacions del vídeo crea el formulari de creació de partners.
- Crea un formulari (`movie-create.php`) per a inserir noves pel·lícules.
- Posa un enllaç al formulari en `movies.php`.
- La pàgina (`movie-create.php`) processarà el formulari i després de validar tots els camps els inserirà a la base de dades.

Cal tenir en compte la separació de la lògica i la presentació.

Exercici pràctic

Modificant i esborrants pel·lícules

ENUNCIAT {:.no_toc .nocount}

- Seguint les indicacions del vídeo crea el formulari d'edició de partners.
- Crea el formulari per a editar pel·lícules.
- Crea el formulari per a esborrar pel·lícules.
- Afig en en $movies un enllaç per a esborrar i editar la película de la fila.

3 Simplificant l'accés a la base de dades

3.1 La classe Database

La classe `Database` serà l'encarregada de gestionar la connexió amb la base de dades, contindrà un mètode estàtic `getConnetion()` que tornarà una instància d'una connexió `PDO`. Els mètodes estàtics són accessibles sense necessitat d'instanciar la classe. Així fent `$pdo = Database::getConnection()` obtindríem una instància de PDO.

Avís: aquesta classe conté diverses males pràctiques que cal evitar. Temps tindrem d'arreglar-ho. De moment l'objectiu és que siga senzilla d'usar.

```php
# src/Database.php

class Database
{
    private PDO $connection;

    private function __construct()
    {
        try {
            $name = "movies";
            $user = "dbuser";
            $pass = "1234";
            $connection = "mysql:host=localhost;charset=utf8";
            $options = [
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                PDO::ATTR_PERSISTENT => true
                ];
```

```
18              $this->connection = new PDO("$connection;dbname=$name", $user,
                    $pass, $options);
19          } catch (PDOException $e) {
20              die ("Eerror en intentar connectar al servidor de base de
                   dades: " . $e->getMessage());
21          } catch (Exception $e) {
22              die("Error en intentar connectar al servidor de base de dades:
                    " . $e->getMessage());
23          }
24      }
25
26      public static function getConnection(): PDO
27      {
28          try {
29              $PDO = new Database();
30          } catch (Exception $e) {
31              die("Error en intentar connectar al servidor de base de dades:
                    " . $e->getMessage());
32          }
33          return $PDO->connection;
34      }
35  }
```

3.2 Arxiu de configuració

En l'exemple anterior les dades de la connexió estan posades directament, el que fa que sempre que vullguen reutilitzar la classe haguem de modificar manualment la classe.

La solució habitual és que totes les dades de configuració que siguen susceptibles de canviar en els diferents entorns (desenvolupament, producció, test, etc.) es separen en un o diversos fitxers de configuració.

Aquests fitxers es poden codificar en PHP o utilitzar un altre format que després puga ser llegit per PHP (Per exemple, ara s'utilitza molt YAML).

Dotenv

Dotenv és una mena d'estàndard de facto per a emmagatzemar la informació sensible de les aplicacions. Mitjançant fitxers `.env` s'estableix la configuració que després es carrega com a variables d'entorn.
El seu origen està en el paquet Ruby dotenv i s'utilitza en diversos frameworks com Symfony i Nodejs.

Per exemple

```php
# config/config.php
$config = [
    'database' => [
        'username' => 'blog',
        'password' => '1234',
        'connection' => 'mysql:host=blog.local;dbname=blog;charset=utf8',
        'options' => [  PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                        PDO::ATTR_PERSISTENT => true ]
    ]
];
```

En format JSON:

```json
{
    "database": {
        "username": "blog",
        "password": "blog",
        "connection": "mysql:host=blog.local;dbname=blog",
        "options": { }
      }
    }
```

3.3 Creació de les entitats

Les entitats són les classes que mapejaran les taules de la base de dades. Tindran un atribut per cada camp de la taula que mapegen. Aquest tipus de classes també poden anomenar-se dominis (*domains*) o DAO (Data Access Objects). Independentment del nom que se li done caldrà tenir en consideració la seua funcionalitat: representar registres d'una taula.

En la majoria dels casos només contindran:

- *getters* i *setters* per accedir als atributs.

- Constructor si cal.
- `__toString` per convertir l'objecte a cadena si cal.
- Etc.

No implementarem lògica de negoci en les entitats.

Les consultes que obtindran dades de la BBDD tornaran un array d'entitats.

Per exemple:

```php
class Book
{
    private $id;
    private $isbn;
    private $title;
    private $author;
    private $stock;
    private $price;

    public function getId(): int
    {
        return $this->id;
    }

    public function getIsbn(): string
    {
        return $this->isbn;
    }

    public function getTitle(): string
    {
        return $this->title;
    }

    public function getAuthor(): string
    {
        return $this->author;
    }

    public function getStock(): int
    {
        return $this->stock;
    }

    public function getCopy(): bool
    {
        if ($this->stock < 1) {
```

```
38              return false;
39          } else {
40              $this->stock--;
41              return true;
42          }
43      }
44
45      public function addCopy()
46      {
47          $this->stock++;
48      }
49
50      public function getPrice(): float
51      {
52          return $this->price;
53      }
54  }
```

Com ja hem vist en l'apartat anterior tant el mètode `fetch` com el mètode `fetchAll` de la classe `PDOStatement` tenen la possibilitat de retornar les dades de la BBDD com a objectes de la classe que li indiquem.

Utilitzarem el paràmetre `PDO::FETCH_CLASS` passant el nom de l'entitat com a segon paràmetre.

```
1  $posts= $stmt->fetchAll(PDO::FETCH_CLASS, 'Book');
```

3.4 Contenidors de serveis

Un contenidor de serveis (o contenidor d'injecció de dependències) simplement és un objecte PHP que gestiona la creació d'instàncies dels serveis (és a dir, dels objectes).

Suposem per exemple que tens una classe PHP senzilla que envia missatges de correu electrònic. Sense un contenidor de serveis, has de crear manualment l'objecte cada vegada que el necessites:

```
1  use Acme\HelloBundle\Mailer;
2
3  $mailer = new Mailer ( 'sendmail');
4  $mailer-> send ('ryan@foobar.net ', ...);
```

Aquest codi és bastant fàcil, ja que la classe imaginària Mailer s'encarrega de configurar el mètode utilitzat per enviar missatges de correu electrònic (per exemple, sendmail, smtp, etc.). Què passa si has d'utilitzar la classe Mailer en un altre punt de l'aplicació? Has de copiar i enganxar el mateix codi en tots els llocs? I si

has de canviar la forma en què s'envien els correus electrònics? Has de buscar en el codi de tota l'aplicació i canviar la mateixa configuració desenes de vegades?

3.5 Model

El model representa la lògica de negocis. S'encarrega d'accedir de forma directa a les dades actuant com a "intermediari" amb la base de datos.

En Symfony, per exemple, no trobareu models, és per això que llegireu que no és un MVC "pur",en canvi disposa de repositoris d'entitats que en la pràtica són homologables als models.

En el nostre MVC tindrem la classe abstracta `Model` que implementarà les operacions habituals amb la base de dades.

```php
abstract class Model
{

  protected string $className;

  protected string $tableName;

  protected PDO $pdo;

  public function __construct(PDO $pdo, string $tableName, string
     $className);

  public function findAll($order = []): array;

  public function find(int $id): Entity;

  public function findBy(array $data = [], $operator = "AND"): array;

  public function findOneBy(array $data = []): ?Entity ;

  public function update(Entity $entity): bool;

  public function save(Entity $entity): bool;

  public function delete(Entity $entity): bool;

  // Rep una senténcia SELECT en paràmetres que seran passats com a un
     array on la clau serà
  // el nom del paràmetre i el valor el valor i torna un array amb el
     resultat.
```

```
28    // per exemple si $sql és "SELECT * FROM movie WHERE title LIKE :text"
          el paràmetre
29    // passat  serà ["text"=>"%Ava%"].
30    public function executeQuery(string $sql, array $parameters = []): array
          ;
31  }
```

3.6 Gestió de les relacions

A l'hora d'obtenir les dades de les entitats relacionades podem optar per dues estratègies: *lazy loading* o *eager loading*.

- *Lazy loading* o càrrega diferida és un patró de disseny que s'utilitza habitualment en programació, sobretot en el disseny i desenvolupament de pàgines web per diferir la inicialització d'un objecte fins al punt en què es necessita.
- *Eager loading* o càrrega immediata és un patró de disseny pel qual una consulta per a un tipus d'entitat també carrega entitats relacionades com a part de la consulta, de manera que no necessitem executar una consulta independent per a entitats relacionades.

En el nostre cas optarem per la càrrega diferida per ser més senzilla. Aprofundirem més en aquestes conceptes quan parlem de *frameworks* com Laravel o Symfony.

3.7 Activitats

Exercici pràctic

Implementació del model

Enunciat

1. Implementa el model seguint les indicacions dels vídeos
2. Implementa els mètodes `delete`, `update` i `executeQuery`. En `update` pots usar `array_map` per a generar la sentència `UPDATE` tenint en compte que la clau primària no pot canviar mai.
3. Fes ús del model sempre que calga interactuar en la base de dades.

```
1  /**
2   * Class UploadedFile
```

```
3   *
4   * Classe que gestiona la pujada de fitxers al servidor mitjançant
        formularis
5   */
6   class UploadedFile
7   {
8       /**
9        * @var array
10       *
11       * Array del fitxer pujat. $_FILES['nom_del_camp_del_formuari']
12       */
13      private $file;
14      /**
15       * @var string
16       *
17       * Nom amb que es guardarà el fitxer.
18       */
19      private $fileName;
20      /**
21       * @var int
22       *
23       * Mida màxima en bytes del fitxer, 0 indica que no hi ha límit.
24       */
25
26      private $maxSize;
27      /**
28       * @var array
29       *
30       * Array amb els MimeType acceptats. Per exemple ['image/jpg', 'image/
             gif', image/png'].
31       * Si l'array és buit  s'accepten tots els tipus.
32       */
33      private $acceptedTypes;
34
35      /**
36       * UploadedFile constructor. Comprova que s'ha pujat un fitxer, si no
             llançarà una excepció
37       * UploadFileNoFileException.
38       * En qualsevol altre error en la pujada llançarà l'excepció
             UploadFileException.
39       *
40       * El paràmetre $inputName rebrà la clau de $_FILES en que s'
             emmagatzemen les dades del
41       * fitxer pujat.
42       *
43       * El paràmetre $maxSize indica la grandària màxima en bytes permesa.
             És opcional,
```

```
44          * serà 0 per defecte.  Si és 0 la grandària és ilimitada.
45          *
46          * El paràmetre $acceptedTypes és opcional. Contindrà els mimetype (
                tipus de fitxers) permesos.
47          * Per defecte estarà buit, en eixe cas es podrà pujar qualsevol tipus
                de fitxer.
48          *
49          * @param string $inputName
50          * @param int $maxSize
51          * @param array $acceptedTypes
52          * @throws UploadFileException
53          * @throws UploadFileNoFileException
54          */
55         public function __construct(string $inputName, int $maxSize = 0,
56             array $acceptedTypes = array());
57
58         /**
59          * @return bool
60          *
61          * Comprova que el fitxer s'haja pujat correctament, que no supera el
62          * limit de grandària i és del tipus indicat.
63          * Si no passa la validació llançarà una excepció.
64          * @throws UploadFileException
65          */
66         public function validate(): bool;
67
68         /**
69          * @return string
70          *
71          * Tornarà el nom del fitxer.
72          */
73         public function getFileName(): string
74         {
75             return $this->fileName;
76         }
77
78         /**
79          * @param string $directory
80          * @param string $fileName
81          * @return bool
82          *
83          * Guarda el fitxer en la ubicació indicada.
84          * Si no s'indica nom es guardarà amb el mateix nom que s'ha penjat.
85          * Si s'indica es guardarà en eixe nom, l'extensió s'obtindrà de forma
                automàtica a partir
86          * del nom del que s'ha pujat.
87          *
```

```php
     * Exemple: $path = '/public/images/', $fileName = 'prova.png'
     *
     * Torna true si s'ha pogut moure la imatge a la ubicació indicada.
     */
    public function save(string $directory, $fileName = ""): bool
    {
        if (!is_uploaded_file($this->file['tmp_name'])) {
            return false;
        }
        // TODO: Implementar el que falta
    }

    /**
     * Extrau del nom de fitxer pujat la seua extensió
     * @return string
     */
    private function getExtension(): string
    {
        $file = $this->file["name"];
        $extensionArray = explode(".", $file);
        $extension = end($extensionArray);
        return $extension ? ".$extension" : "";
    }
}
```

Exercici pràctic

La classe FileUpload

Enunciat Fes una ullada a la classe FileUpload anterior.

1. A partir de la documentació en PHPDoc implementa els mètodes que falten de la classe `FileUpload`.
2. Implementa també les excepcions personalitzades.
3. Fes ús de la classe per a gestionar la pujada d'imatges en pel·lícules, limitant-les al format jpg i a la grandària de 300KB.
4. En Handling file uploads teniu informació addicinal. No cal que es cree el directori si no existeix. Anem a suposar que ja existeix.

Exercici pràctic

Gèneres

Enunciat

1. Implementa el model per a la taula `Genre`.
2. En `Movie` afig la propietat `$genre_id` que serà la clau aliena a la taula `Genere` i crea els mètodes `Movie::setGenreId()` i `Movie::getGenreId()` per a accedir a la propietat.
3. Modifica el formulari de creació de pel·lícules perquè es puga triar el gènere d'una llista desplegable.
4. En el llistat de pel·lícules (`movies.php`) afig una columna que mostre el `id` de gènere.
5. Modifica `Model::save()` perquè assigne l'últim id inserit a l'element Entity.

Exercici pràctic

Obtenint el gènere

Enunciat

1. Implementa el mètode `MovieModel::getGenre()` de forma que en rebre l'id d'un gènere et torne l'objecte gènere relacionat.
2. Utilitza el mètode anterior perquè en `movies.php` aparega el nom del gènere enlloc del seu `id`.

4 Transaccions

4.1 Introducció

Una transacció en un Sistema de Gestió de Bases de Dades és un conjunt d'ordres que s'executen formant una unitat de treball, és a dir, en forma indivisible o atòmica.

Un SGBD es diu transaccional si és capaç de mantenir la integritat de dades, fent que aquestes transaccions no puguin finalitzar en un estat intermedi. Quan per alguna causa el sistema ha de cancel·lar la transacció, comença a desfer les ordres executades fins a deixar la base de dades en el seu estat inicial (anomenat punt d'integritat), com si l'ordre de la transacció mai s'hagués realitzat. Una transacció ha de

comptar amb ACID (un acrònim anglès) que vol dir: Atomicidad, consistència, aïllament i durabilitat.

4.2 Transaccions en PDO

Com ja hem comentat, no tots els SGBD són transaccionals. Però també es possible que tot i que el SGBD ho siga el motor d'emmagatzemat no ho suporte. Per la qual cosa si necessites utilitzar transaccions hauràs d'assegurar-se que estiguen suportades pel motor d'emmagatzematge que gestiona les teves taules en MySQL. Per exemple el motor MyISAM no les suporta.

Per defecte PDO treballa en mode autocommit. Confirma automàticament cada sentència que executa el servidor.

Per treballar amb transaccions, PDO incorpora tres mètodes:

- `beginTransaction`. Deshabilita la manera autocommit i comença una nova transacció, que finalitzarà quan executis un dels dos mètodes
  següents.
- `commit`. Confirma la transacció actual.
- `rollback`. Reverteix els canvis duts a terme a la transacció actual.

Un cop executat un `commit` o `rollback`, es tornarà al mode de confirmació automàtica.

Si utilitzem un motor que no suporta transaccions PDO no mostrarà cap error, simplement serà incapaç de realitzar un `rollback` si fora necessari.

Esquema bàsic d'ús de transaccions amb PDO

```
1     #Esquema bàsic d'ús de PDO
2     try{
3         $pdo->beginTransaction();
4         $pdo->exec('DELETE ...');
5         $pdo->exec('UPDATE ...');
6         $pdo->commit();
7     }
8     catch(PDOException $exception) {
9         $pdo->rollback();
10    }
```

Exemple

```php
    $mbd = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2',
             array(PDO::ATTR_PERSISTENT => true));
        echo "Conectado\n";
    } catch (Exception $e) {
        die("No se pudo conectar: " . $e->getMessage());
    }

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

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

    } catch (Exception $e) {
        $mbd->rollBack();
        echo "Fallo: " . $e->getMessage();
    }
```

Exercici pràctic

Transaccions

Enunciat Implementa una transacció de forma que les operacions d'inserció de pel·lícules en la base de dades i d'actualització del número de gèneres s'execute de forma única (com si fos una).

5 Paginació

La paginació és el procés de dividir un document en pàgines. En el nostre cas, limitar el nombre de registres una taula que volem mostrar en cada pàgina.

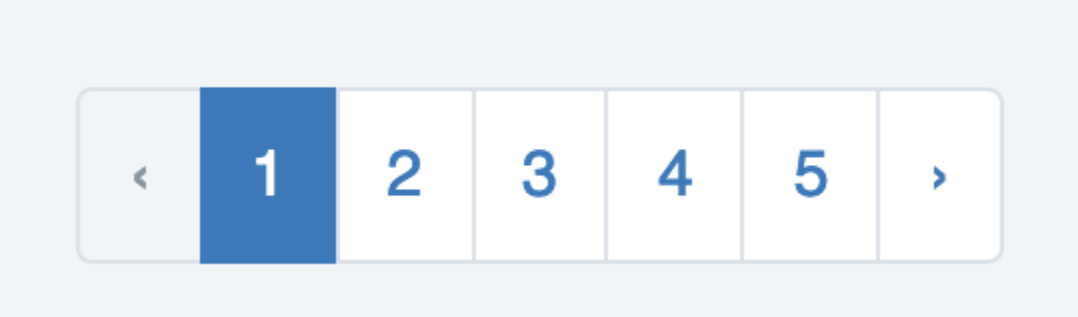

Figure 3: Paginació

En MySQL disposem de les claúsules LIMIT i OFFSET per poder implementar la paginació.

Exemple:

```
SELECT * FROM post LIMIT 4 OFFSET 0
```

La consulta tornaria 4 registres a partir del primer (OFFSET 0).

Hi ha una sintaxi alternativa en la que els valors es separem per coma i s'omet la clau OFFSET:

```
SELECT * FROM post LIMIT 0, 4
```

En aquest cas l'offset es situa després de límit.

A l'hora de realitzar la consulta ho farem mitjançant consultes preparades, passant com a paràmetres l'offset i el límit.

```
$stmt=→$connprepare(
"SELECT * FROM movie
 LIMIT :limit
 OFFSET :offset");→
$stmtbindValue(':limit', 4);→
$stmtbindValue(':offset', 0);→
$stmtexecute();
```

5.1 Obtenir les dades paginades

Per a implementar la paginació necessitarem saber:

- El límit (quants registres mostrarem).

- L'offset (a partir de quin registre mostrem).

Això ho calcularem a partir de:

- La grandària de la pàgina (límit).
- El número de pàgina actual (per a calcular l'offset).

Per conèixer el número total de pàgines caldrà saber el número total de registres.

```
$numberOfRecordsPerPage = 4;

// el número de pàgina es sol passar en un paràmetre del _querystring_.
$currentPage = filter_input(INPUT_GET, "page", FILTER_VALIDATE_INT);

// si $currentPage és false o null després del filter_input assignarem
// el valor 1 per defecte.

if (empty($currentPage))
    $currentPage = 1;

// Calculem el offset
// si $currentPage = 1, $offset = 0
// si $currentPage = 2, $offset = 4
$offset = ($currentPage-1)*$numberOfRecordsPerPage;

// el límit el determina la grandària de pàgina
$limit = $numberOfRecordsPerPage;

$stmt=→$connprepare(
"SELECT * FROM movie
 LIMIT :limit
 OFFSET :offset");→
$stmtbindValue(':limit', $limit);→
$stmtbindValue(':offset', $offset);→
$stmtexecute();
```

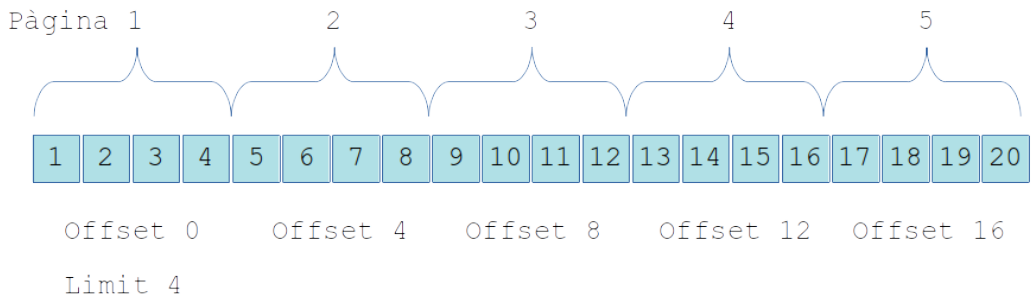

Figure 4: Paginació

5.2 Navegar entre les pàgines

Obtenir les dades paginades és el primer pas. Després caldrà implementar el típic navegador.

Teniu un exemple en aquest tutorial How to implement pagination in MySQL

Exercici pràctic

Paginació

Enunciat

1. Modifica `index.php`de forma que sols mostre les 8 pel·lícules més noves.
2. Implementa un paginador en `movies.php` de forma que mostre 10 pel·lícules per pàgina i pugam navegar entre pàgines.
3. Implementa `Model::findAllPaginated()` amb 3 paràmetres opcionals $currentPage que rebrà la pàgina actual, per defecte serà 1; $numberOfRecords que serà la grandària de la pàgina, per defecte serà 10; i $order que funcionarà com en `Model::findAll()`. [Opcional].

6 Procediments emmagatzemats

6.1 Introducció

Un procediment emmagatzemat és un programa (o procediment) emmagatzemat dins d'una base de dades. Estan escrits en llenguatges d'alt nivell (normalment propietaris, és a dir, cada SGBD implementa el seu) i contenen sentències d'SQL.

El principal avantatge d'un procediment emmagatzemat és que quan s'executa, com a conseqüència d'una petició d'un usuari, ho fa directament en la base de dades i per tant és més ràpid processant dades que si el mateix procés es realitzés fora de la base de dades (probablement en un altre servidor), ja que estalvia la transmissió de les dades que tracta.

6.2 Implementació

La sentència SELECT següent retorna totes les files de la taula `movie` la nostra base de dades:

```
SELECT
    *
FROM
    movie
ORDER BY title;
```

Si voleu desar aquesta consulta al servidor de base de dades per executar-la més endavant, una manera de fer-ho és utilitzar un procediment emmagatzemat.

La sentència CREATE PROCEDURE següent crea un nou procediment emmagatzemat que completa la consulta anterior:

```
DELIMITER $$

CREATE PROCEDURE GetMovies()
    BEGIN
    SELECT
        *
    FROM
    movie
    ORDER BY title;
END$$
DELIMITER ;
```

Per definició, un procediment emmagatzemat és un conjunt d'instruccions SQL emmagatzemades dins del servidor MySQL. En aquest exemple, acabem de crear un procediment emmagatzemat amb el nom `GetMovies()`.

Un cop desat el procediment emmagatzemat, el podeu invocar mitjançant la sentència CALL:

```
1 CALL GetMovies();
```

I la sentència retorna el mateix resultat que la consulta.

La primera vegada que invoqueu un procediment emmagatzemat, MySQL busca el nom al catàleg de la base de dades, compila el codi del procediment emmagatzemat, el col·loca en una àrea de memòria coneguda com a memòria cau i executa el procediment emmagatzemat.

Si torneu a invocar el mateix procediment emmagatzemat a la mateixa sessió, MySQL només executa el procediment emmagatzemat des de la memòria cau sense haver de recompilar-lo.

Un procediment emmagatzemat pot tenir paràmetres perquè pugueu passar-hi valors i recuperar-ne el resultat. Per exemple, podeu tenir un procediment emmagatzemat que retorne les pel·lícules per gènere. En aquest cas, el gènere seria un paràmetre del procediment emmagatzemat.

Un procediment emmagatzemat pot contenir declaracions de flux de control, com ara IF, CASE i LOOP que li permeten implementar el codi en la forma de procediment.

Un procediment emmagatzemat pot cridar a altres procediments emmagatzemats o funcions emmagatzemades, cosa que permet modular el codi.

6.3 Canvi temporal del delimitador

Com que el delimitador de sentències per defecte en MySQL és el punt i coma (;). Si intenteu crear un procediment emmagatzemant amb diverses sentències el client MySQL interpretarà que cada sentència és diferent i podeu tenir problemes.

La forma d'evitar aquest problema és canviar temporalment el delimitador.

```
1 -- Canvie el delimitador
2 DELIMITER //
3
4 -- Ací posaríem la declaració del procediment emmagatzemant
5 CREATE PROCEDURE  ...
6  BEGIN
7     SELECT * ...;
```

```
    SELECT * ...;
 END

-- Tornem al delimitador per defecte
DELIMITER ;
```

6.4 Executar procediments emmagatzemats des de PHP

La forma d'executar procediments emmagatzemats des de PHP és molt semblant a executar qualsevol consulta:

En aquest exemple es crida un procediment emmagatzemant sense paràmetres:

```
$sentencia = $mbd->prepare("CALL sp_update_counters()");
// llamar al procedimiento almacenado
$sentencia->execute();
```

Exercici pràctic

Procediment emmagatzemat

Enunciat

1. Crea un procediment emmagatzemant anomenat `sp_update_number_of_movies()` que actualitze els contadors de pel·lícues de gènere. Aquest procediment serà excutat regularment perquè cada gènere tinga el número de pel·lícules del gènere actualitat.
2. Crea en el projecte la pàgina `tasks.php` que incloga l'execució del procediment emmagatzemant.

6.5 Webgrafia

- The PHP Group. *Sentencias preparadas y procedimientos almacenados* [En línia]. Manual de PHP [Data de consulta: 20 de novembre de 2020]. Disponible en https://www.php.net/manual/es/pdo.prepared-statements.php
- MySQLtutorial.org. *Introduction to MySQL Strored Procedures* [En línia]. MySQLTutorial [Data de consulta: 20 de novembre de 2020]. Disponible en https://www.mysqltutorial.org/introduction-

to-sql-stored-procedures.aspx
— layout: default parent: 5. Accés a dades amb PHP PDO nav_order: 7 has_children: false —

7 Triggers

Els disparadors (triggers) són procediments emmagatzemats a la base de dades que s'executen automàticament quan es du a terme una operació INSERT, DELETE o UPDATE sobre alguna taula en concret i permeten als usuaris executar funcions introduïdes per altres usuaris sense conèixer-ne la implementació.

No es poden dur a terme en una operació SELECT.

Aquesta acció podria servir per fer una comprovació de consistència en un conjunt de valors per ser inserits, en el format de les dades abans de ser introduïdes, en una modificació en una taula o en una modificació d'un conjunt de files.

L'estàndard SQL defineix dos tipus d'activadors: activadors a nivell de fila i activadors a nivell d'instruccions.

- S'activa un activador a nivell de fila per a cada fila que s'insereix, s'actualitza o se suprimeix. Per exemple, si una taula té 100 files inserides, actualitzades o suprimides, el disparador s'invoca automàticament 100 vegades per a les 100 files afectades.
- Un activador a nivell de sentència s'executa una vegada per a cada transacció independentment del nombre de files inserides, actualitzades o suprimides.

MySQL només admet activadors a nivell de fila. No admet els activadors a nivell de sentència.

En el Tutorial de Triggers MySQL teniu més informació.

En el nostre cas implementarem el següent disparador que actualitzarà el contador de pel·lícules de gènere quan hi haja una modificació en el gènere d'una pel·lícula.

Aquesta és la sintaxi d'un trigger BEFORE UPDATE

```
1 CREATE TRIGGER trigger_name
2 BEFORE UPDATE
3 ON table_name FOR EACH ROW
4 trigger_body
```

En aquest exemple el trigger executarà després (AFTER UPDATE) d'actualitzar una pe·lícula:

```
DELIMITER $$

CREATE TRIGGER after_movie_update
AFTER UPDATE
ON movie FOR EACH ROW
BEGIN
    -- Si hi ha un canvi en el género
    IF new.genre_id <> old.genre_id THEN
        -- Restem 1 al contador del género antic
        UPDATE genre SET number_of_movies = number_of_movies -1 WHERE id =
            old.genre_id;
        -- Sumem 1 al contador del género nou
        UPDATE genre SET number_of_movies = number_of_movies +1 WHERE id =
            new.genre_id;
    END IF;
END$$

DELIMITER ;
```

Trigger

Enunciat

1. Implementa un *trigger* de forma que en actualitzar una pel·lícula es comprove si ha canviat el gènere seleccionat. En cas afirmatiu que decremente el contador de pel·lícules en el gènere anterior i augment en el nou.
2. Implementa un *trigger* actualitze el contador de pel·lícules en esborrar una pel·lícula.

Crea una carpeta anomenada `data` en el projecte i guarda en ella una exportació de la base de dades.

8 Errors freqüents

8.1 Problemes amb $releaseDate

La funció `fechAll()` amb el mode `FETCH_CLASS` torna un array d'objectes de la classe que li hem passat com a 2n paràmetre. El que fa és assignar cada camp de la taula a una propietat de la classe, *si la propietat no existeix la crea*.

Al treballar en tipat estricte `declare(strict_types=1)`, si el tipus d'una propietat no és escalar (tipus bàsic), la funció `fetchAll()` amb el mode `FECHT_CLASS` falla.

Eixe és el problema si usem **`private`** `DateTime $release_date`, que com la data la intenta guardar en `string` i la propietat és de tipus `DateTime` mostra un error.

La solució més elegant que he trobat i que ens permet usar el mode `FETCH_PROPS_LATE` en el `fechtAll()` es tractar d'aprofitar que quan recuperem dades d'una taula amb `FETCH_CLASS` si les propietats que no eixisteixen en la classe les crea.

Això dispara el mètode màgic `__set($name, $value)` i perment que puguem llegir el valor i convertir-lo en `DateTime` sobre la propietat $releaseDate.

Per això hem de llevar la propietat `$release_date` i gestionar-ho en el `__set` de la següent manera:

```php
private int $id;
private string $title;
private string $overview;
// private string $release_date;
private DateTime $releaseDate;
private float $starsRating;
private ?string $tagline;
private string $poster;

public function __set(string $name , $value) {
    switch ($name) {
        case "release_date":
            $this->releaseDate = DateTime::createFromFormat("Y-m-d",
                $value);
            break;
    }
}
```

El constructor ja no serà necessari.

8.2 Valors que poden ser null

Es possible que alguns valor en una taula puguen ser **`null`** (no ser obligatoris). Recordeu que **`null`** en PHP és un tipus així que també, a consequència del tipat estricte, fallarà.

La solució la podeu veure en l'exemple anterior, el símbol `?` india que la propietat `$tagline` potser `string` o **`null`**.

Òbviament, si un camp pot ser **`null`**, s'hauria d'indicar en la base de dades i no hauria de ser obligatori en els formularis.

8.3 Els fitxers es pugen quan hi ha errors

En els formularis d'inserció i edició si hi ha un error de validació en algun camp i s'ha pujat una imatge, la imatge es puja i es guarda igualment.

8.4 L'entitat `movie` o `partner` no es carrega

Hi ha alguns casos en que l'entitat `movie` o `partner` no es carrega adequadament si s'envia el formulari d'edició amb errors. El motiu és perquè per a carregar-se depén el *querystring*.

Per exemple:

```php
$id = filter_input(INPUT_GET, 'id', FILTER_VALIDATE_INT);
if (empty($id)) {
    $errors[] = "404 Not found";
} else {
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $pdo->prepare('SELECT * FROM movie WHERE id=:id');
    $stmt->bindValue("id", $id, PDO::PARAM_INT);
    $stmt->execute();
    $stmt->setFetchMode(PDO::FETCH_CLASS, Movie::class);
    $movie = $stmt->fetch();
}
```

En el vídeo a mi en funcionava per casualitat (ho explicaré després) perquè per un error el formulari s'enviava a la url amb querystring, per exemple, `http://00-vicent.local/movies-edit.php?id=245`.

8.5 URL estranya

El tema de la URL estranya està motivada perquè en alguns sistemes, en el meu per exemple, puc fer açò:

```php
<? $_SERVER["PHP_SELF"] ?>
```

Hem permet usat `<?` en lloc de `<?php`. El motiu el desconec. Per això a mi no m'escrivia res, m'havia oblidat de posar el = i. En el cas de Pepe i David no, perquè el seu sistema no ho interpreta com codi PHP i suposa que es HTML, per això en enviar el formulari els eixia la URL estranya.

8.6 Validació i separació de lògica i presentació

Tant la validació de les dades rebudes del formulari com la separació de la lògica i la presentació són bones pràctiques que s'han de fer sempre.

8.7 Paràmetres

Els paràmetres de les consultes preparades no es poden usar com si foren les marques `%s` de `sprintf()`, sols es poden aplicar en el WHERE o el HAVING de la forma que hem vist.

```
1  $stmt->bindValue("id","$id", PDO::PARAM_INT);
```

En el codi anterior sobrnm les cometes de `"$id"` ja que $id és enter i el paràmetre també ho ha de ser, en este cas s'està passant un enter a string perquè `bindValue` el passe internament a enter.

L'error bé perquè en el filtre fèiem `"%$text%"`, ahí és correcte perquè estem fent una expansió de la variable (una mena de concatenació), ja que és el paràmetre d'un LIKE.