PDO

Principe

- PDO – PHP Data Objects
- Couche d'accès aux bases de données fournissant une méthode uniforme pour accéder à plusieurs bases de données.

- On n'utilise plus mysql et mysqli

Avantages

- couche d’abstraction
- syntaxe orientée objet
- prise en charge des requêtes préparées
- meilleure gestion des exceptions
- APIs sécurisées et réutilisables
- prise en charge de toutes les bases de données populaires

Database support

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

- Pour voir la liste des drivers

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

Connection

```
                Database Type
$DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
Database Handle      Database Specific Connection String
```

```php
try {
  # MS SQL Server and Sybase with PDO_DBLIB
  $DBH = new PDO("mssql:host=$host;dbname=$dbname", $user, $pass);
  $DBH = new PDO("sybase:host=$host;dbname=$dbname", $user, $pass);

  # MySQL with PDO_MYSQL
  $DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);

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

Insert et update

- On peut utiliser la méthode exec() mais c'est dangereux sur l'injection SQL

```php
$DBH->exec('INSERT INTO xxx');
```

Requête SQL classique (vulnérable à l'injection SQL) :

```sql
"SELECT * FROM users WHERE username = '" + user_input + "';"
```

Si `user_input` est `"' OR '1'='1"`, la requête deviendrait :

```sql
"SELECT * FROM users WHERE username = '' OR '1'='1';"
```

Cela pourrait retourner tous les utilisateurs dans la table, car `'1'='1'` est toujours vrai.

Insert and update


```
# unnamed placeholders
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
```

```
# assign variables to each place holder, indexed 1-3
$STH->bindParam(1, $name);
$STH->bindParam(2, $addr);
$STH->bindParam(3, $city);

# insert one row
$name = "Daniel"
$addr = "1 Wicked Way";
$city = "Arlington Heights";
$STH->execute();

# insert another row with different values
$name = "Steve"
$addr = "5 Circle Drive";
$city = "Schaumburg";
$STH->execute();
```

- Requête préparé
  - Précompilation
  - Echappement auto des caractères

Selection de données

- On utilise `->fetch()` et on a plusieurs options

`PDO::FETCH_ASSOC`: returns an array indexed by column name.
`PDO::FETCH_BOTH` (default): returns an array indexed by both column name and number.
`PDO::FETCH_BOUND`: assigns the values of your columns to the variables set with the `->bindColumn()` method.
`PDO::FETCH_CLASS`: assigns the values of your columns to properties of the named class. It will create the properties if matching properties do not exist.
`PDO::FETCH_INTO`: updates an existing instance of the named class.
`PDO::FETCH_LAZY`: combines `PDO::FETCH_BOTH`/`PDO::FETCH_OBJ`, creating the object variable names as they are used.
`PDO::FETCH_NUM`: returns an array indexed by column number.
`PDO::FETCH_OBJ`: returns an anonymous object with property names that correspond to the column names.

Autres méthodes

- `$DBH->lastInsertId();` : retourne la dernière valeur Auto Increment
- `$STH->rowCount();` : retourne le nombre de ligne affectée par l'opération