PHP - SQL

m2iformation.fr

TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

PHP - SQL

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction

Connection à une base de données

Il y a plusieurs APIs PHP pour accéder à une base de données MySQL.
Les utilisateurs peuvent choisir entre les extensions:

- mysqli
- PDO (PHP Data Objects)

Qu'est-ce qu'une API ?

- Une interface de programmation d'application, définit les **classes**, les **méthodes**, les **fonctions** et les **variables** dont l'application a besoin pour réaliser les tâches désirées
- Dans le cas des applications PHP qui ont besoin de communiquer avec des bases de données, les APIs nécessaires sont habituellement exposées via des extensions PHP

Qu'est-ce qu'un connecteur ?

- Le terme connecteur se réfère à la partie du programme qui autorise votre application à se connecter au serveur de base de données
- MySQL fournit des connecteurs pour bons nombres de langages, incluant PHP

Qu'est-ce qu'un driver ?

- Un driver est une partie de programme dont le but est de communiquer avec un type spécifique de serveur de base de données.
- Le driver peut également appeler une bibliothèque, comme la bibliothèque cliente MySQL ou le driver natif MySQL. Ces bibliothèques implémentent le protocole bas niveau utilisé pour communiquer avec le serveur de base de données MySQL.

Qu'est-ce qu'une extension ?

- Dans la documentation PHP, vous trouverez un autre terme: *extension*
- Le code PHP est constitué d'un cœur, avec des extensions optionnelles permettant d'étendre les fonctionnalités du cœur
- Les extensions PHP relatives aux bases de données, comme l'extension mysqli, est implémentée en utilisant le framework des extensions PHP
- Une extension n'expose pas forcément une API au programmeur

TOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

PDO

Présentation

- `PDO` fournit une interface d'abstraction à l'accès de données ce qui permet d'utiliser les mêmes fonctions pour exécuter des requêtes ou récupérer les données quel que soit la base de données utilisée
- `PDO` est fournit directement avec PHP

Activer PDO

- PDO et le pilote PDO_SQLITE sont activés par défaut
- le fichier `php.ini` doit être mis à jour afin de charger l'extension PDO automatiquement lorsque PHP entre en fonctionnement
- Pour activer PHP MySQL, dé-commenter la ligne:

```
extension=php_pdo_mysql.dll
```

Connexion à une base de données

- Une connexion est établie à l'aide d'une instance de la classe `PDO`
- Le constructeur de PDO attend plusieurs paramètres:
  1. Source de données
  2. username
  3. password
  4. options

```php
$db = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
```

Fermer une connexion

Pour fermer une connexion à une base de données, il suffit d'attribuer la valeur `null` à l'instance créée

```php
$db = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
$statement = $db->query('SELECT * FROM foo');

$statement = null;
$db = null;
```

Exécuter une requête SQL

- Une fois l'objet PDO instancié, il est possible d'utiliser la méthode `exec` pour exécuter une requête SQL
- Cette méthode exécute la requête et renvoie le nombre de lignes affectées
- On l'utilise généralement pour les requêtes sans paramètres qui ne font pas de lecture

```php
$count = $db->exec("DELETE FROM fruit");
```

Préparer une requête SQL

- L'objet PDO possède une méthode pour préparer une requête, la méthode `prepare` renvoie un objet `PDOStatement`
- L'objet `PDOStatement` permet d'utiliser le binding de paramètres, ce mécanisme permet de se protéger des injections SQL
- Les paramètres seront préfixé du caractère `:` dans la requête

```php
$sql = "SELECT firstname, lastname FROM users WHERE id=:id";
$statement = $db->prepare($sql);
$statement->execute(['id' => 1]);
```

Liaison de paramètres (1/2)

Lorsque l'on instancie un objet `PDOStatement` avec la méthode `prepare` de la connexion PDO, nous avons la possibilité d'utiliser différentes techniques pour binder nos paramètres à la requête :

1. Lorsque l'on utilise la méthode `execute` sur l'objet `PDOStatement`, il suffit de passer un tableau associatif :

```php
$sql = "SELECT firstname, lastname FROM users WHERE age=:age AND LENGTH(firstname) > :length";
$statement = $db->prepare($sql);
$statement->execute([ 'age' => 18 ,'length' => 4]);
```

Liaison de paramètres (2/2)

2. Il est également possible d'utiliser la méthode `bindParam` pour binder chaque paramètre en précisant son type

```php
$couleur = "rouge";
$st = $db->prepare('SELECT nom, couleur, calories
    FROM fruit
    WHERE AND couleur = :couleur');
$st->bindParam('couleur', $calories, PDO::PARAM_STR);
$st->execute();
```

Lire un jeu de résultat

- La méthode `fetch` récupère la ligne suivante d'un jeu de résultats PDO
- Cette méthode prend en paramètre un mode qui contrôle comment sera renvoyé le prochain enregistrement
- Par défaut cette valeur vaut `PDO::FETCH_BOTH`
- La plupart du temps, nous utiliserons `PDO::FETCH_ASSOC` pour récupérer l'enregistrement sous forme de tableau associatif

```php
$result = $sth->fetch(PDO::FETCH_ASSOC);
```

Lire tous les résultats

- L'objet `PDOStatement` fournit également une méthode `fetchAll()` qui permet de récupérer l'ensemble des enregistrements d'un seul coup
- Son fonctionnement est très similaire à celui de `fetch`

```php
$sth = $dbh->prepare("SELECT nom, couleur FROM fruit");
$sth->execute();

$result = $sth->fetchAll(PDO::FETCH_ASSOC);
print_r($result);
```

Récupérer un id généré en base

- `lastInsertId` sur l'objet PDO permet de récupérer le dernier id généré par la base de données
- ⚠️ lors de l'utilisation d'une transaction, il faudra exécuter la méthode avant de commit la transaction
- ⚠️ en cas d'insertion multiple dans la même requête, l'id renvoyé sera le premier qui a été généré (MySQL, MariaDB)

```php
$stmt = $db->prepare("INSERT INTO user (name, email) VALUES(:name, :email)");
$stmt->execute(['name' => 'toto', 'email' => 'toto@titi.fr']);
print $db->lastInsertId();
```

Méthodes supplémentaires

La méthode `rowCount` de l'objet PDOStatement permet de récupérer le nombre de lignes affectées par une requête

```
$del = $db->prepare('DELETE FROM fruit');
$del->execute();
$count = $del->rowCount();
print "Effacement de $count lignes.\n";
```

La méthode `setFetchMode` permet de définir le mode par défaut

```
$stmt = $db->query('SELECT name, color, calories FROM fruit');
$stmt->setFetchMode(PDO::FETCH_ASSOC);
```

Transactions

Définition

- Les transactions offrent 4 fonctionnalités majeures : **A**tomicité, **C**onsistance, **I**solation et **D**urabilité (ACID)
- En d'autres termes, n'importe quel travail mené à bien dans une transaction, même s'il est effectué par étapes, est garanti d'être appliqué à la base de données sans risque, et sans interférence pour les autres connexions, quand il est validé
- Les transactions sont typiquement implémentées pour appliquer toutes vos modifications en une seule fois

Démarrer une transaction

- Tous les mécanismes liés à une transaction sont implémentés directement dans l'objet PDO qui est utilisé pour la connexion
- Pour démarrer une transaction, il suffit d'utiliser la méthode `beginTransaction`

```php
$dbh = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2', array(PDO::ATTR_PERSISTENT => true));
$dbh->beginTransaction();
```

Valider ou annuler une transaction

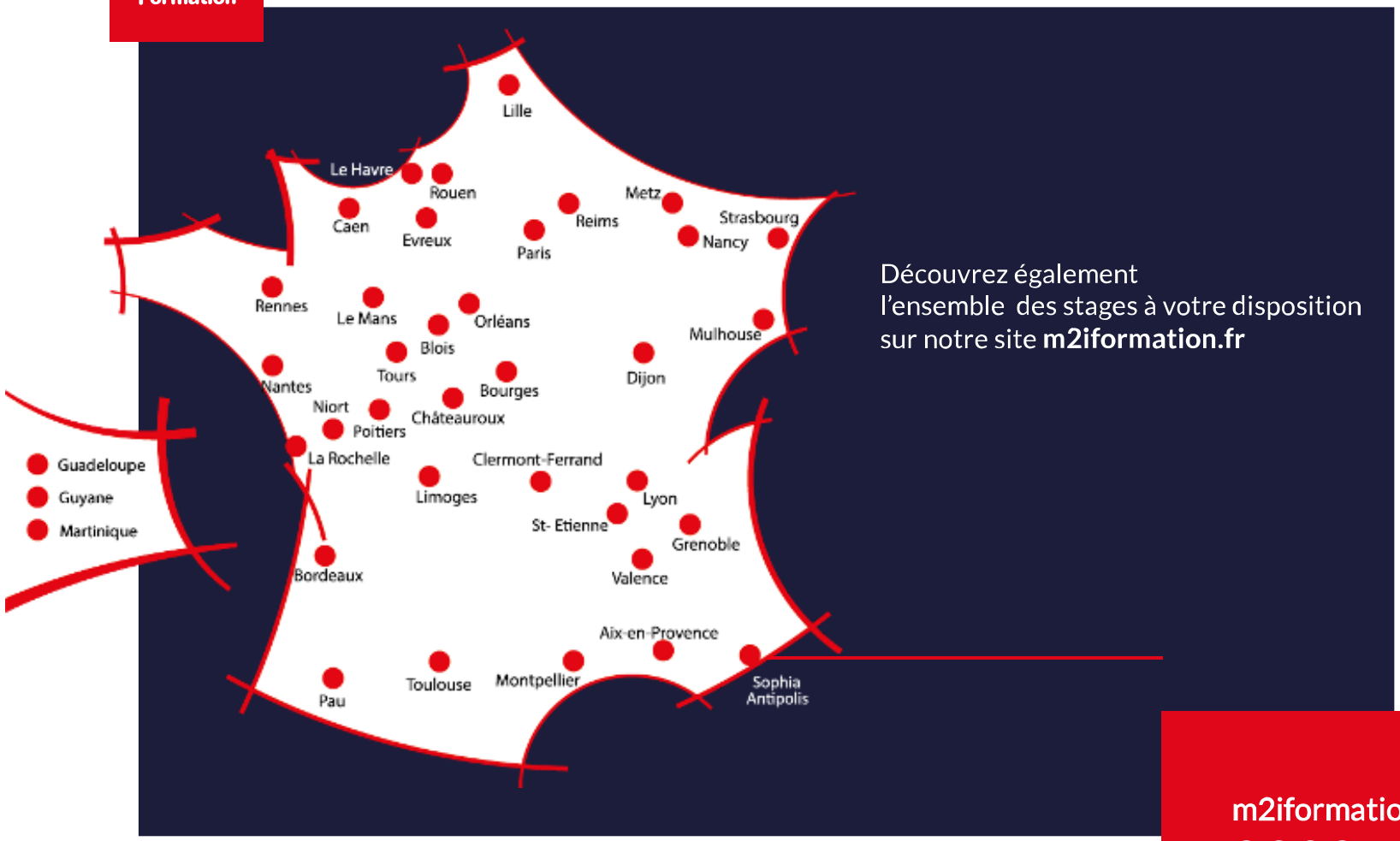
Pour valider une transaction, il suffit d'utiliser la méthode `commit` tandis que pour l'annuler, on utilisera la méthode `rollback`

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

} catch (Exception $e) {
  $dbh->rollBack();
}
```

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr