Introduction à PHP

Alternatives à MySQL

---

monnerat@u-pec.fr

Updated: 2017/05/01

IUT de Fontainebleau

Sommaire

Introduction

PHP et SGBDR

PHP s'interfacent avec a peu près tous les SGBD relationnels.

- Avec serveur externe :
  - API dédiée pour `MySQL` (déjà vue), `PostgreSQL`, `Oracle`.
  - API unique via `ODBC` qui permet d'attaquer la plupart des serveurs.
- Solution intégrée (pas de serveur de bd) à PHP (depuis la version 5) avec `SQLite` : utilisation de fichiers locaux.

Il existe depuis PHP5 une couche abstraite pour les communications avec les sgbd : `PDO`.

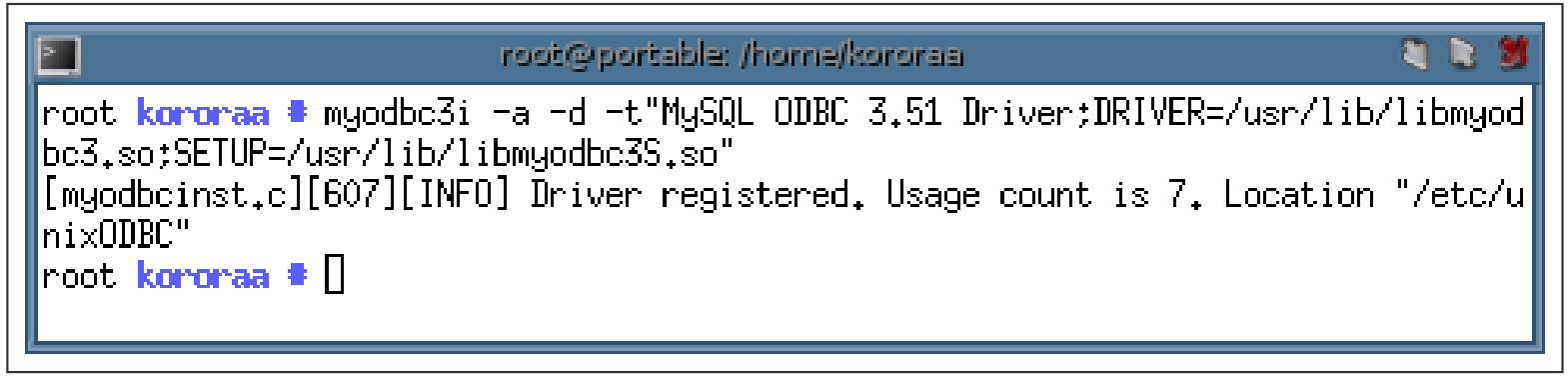
- Nécessite des drivers spécifiques à chaque sgbd.
- Api objet.

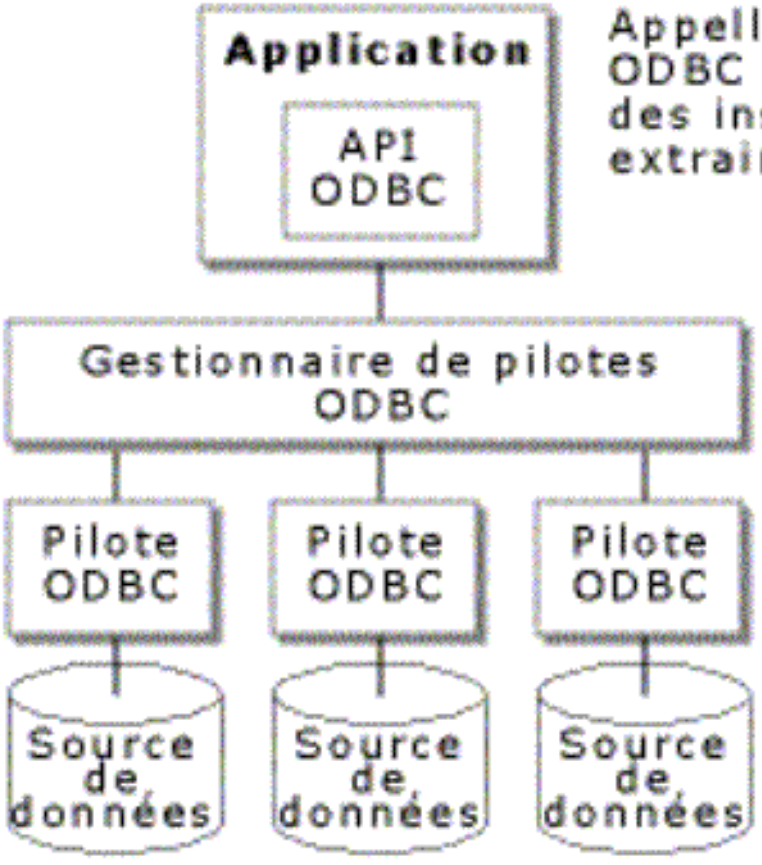
ODBC

- Open Database Connectivity (ODBC) est une puissante API développée par Microsoft pour que les développeurs puissent assurer l'interface avec n'importe quelle base de données compatible (notamment Microsoft SQL Server, Oracle, MySql et bien d'autres encore).

- Open Database Connectivity (ODBC) est une puissante API développée par Microsoft pour que les développeurs puissent assurer l'interface avec n'importe quelle base de données compatible (notamment Microsoft SQL Server, Oracle, MySql et bien d'autres encore).
- Il existe des drivers libres pour beaucoup de SGBDR : par exemple, `MyODBC` pour MySQL.

- Open Database Connectivity (ODBC) est une puissante API développée par Microsoft pour que les développeurs puissent assurer l'interface avec n'importe quelle base de données compatible (notamment Microsoft SQL Server, Oracle, MySql et bien d'autres encore).
- Il existe des drivers libres pour beaucoup de SGBDR : par exemple, `MyODBC` pour MySQL.





Appelle les fonctions de l'API ODBC pour soumettre des instructions SQL et extraire des résultats

Charge les pilotes ODBC pour les applications, transmet les demandes au pilote et les résultats à l'application

Traite les appels de fonctions ODBC, soumet les demandes SQL à une source de données spécifique et à une application

Traite les demandes du pilote et renvoie les résultats au pilote

Connexion

Pour utiliser une connexion ODBC à un serveur de base de données, il est nécessaire de configurer la source de données. Deux méthodes :

- Définir un nom de source de données (DSN Système)
- Définir une chaîne de connexion

DSN Système

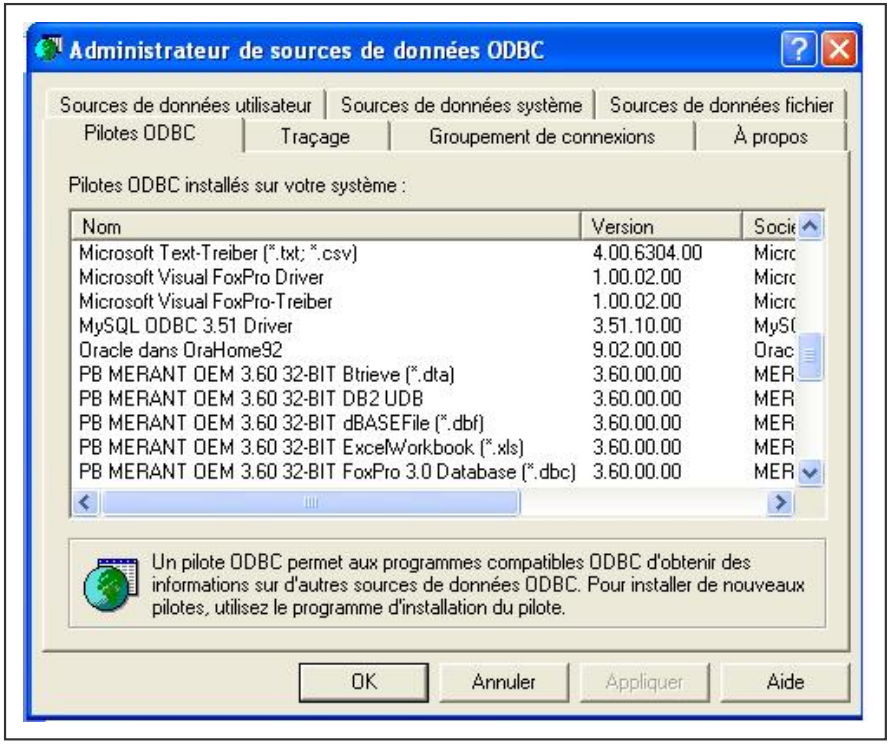
- Un DSN système permet de se connecter à un serveur de base de données en utilisant un alias plutôt que de devoir saisir une longue chaîne de connexion.

DSN Système

- Un DSN système permet de se connecter à un serveur de base de données en utilisant un alias plutôt que de devoir saisir une longue chaîne de connexion.
- Sous Windows XP, Pour lancer l'assistant de création d'un DSN : Démarrer > Panneau de configuration > Outils d'administration > Sources de données (ODBC)

DSN Système

- Un DSN système permet de se connecter à un serveur de base de données en utilisant un alias plutôt que de devoir saisir une longue chaîne de connexion.
- Sous Windows XP, Pour lancer l'assistant de création d'un DSN : Démarrer > Panneau de configuration > Outils d'administration > Sources de données (ODBC)



Chaîne de connexion

Pour se connecter sans définir de DSN, il suffit d'utiliser une chaîne de connexion.

```
$connection_string = "DRIVER={MYSQL ODBC 3.51 Driver};
     SERVER=localhost;DATABASE=tp3;";
```

Connexion à la source ODBC

```
resource odbc_connect ( string $dsn , string $user ,
  string $password    [, int $cursor_type ] )
```

retourne un identifiant de connexion ODBC ou 0.

```
$dsn="sqlserver";
$username="banque";
$password="";
$sqlconnect=odbc_connect($dsn,$username,$password);
```

La chaîne dsn peut être une chaîne de connexion.

```
$connection_string ="DRIVER={MySQL ODBC 3.51 Driver};
                     SERVER=localhost;DATABASE tp3;";
$username="root";
$password="";
```

Requête

```
resource odbc_exec ( resource $connection_id ,
          string $query_string)
```

retourne un identifiant de résultat ODBC ou FALSE en cas d'erreur.

```
<?php
$sqlquery="SELECT * FROM Employe;";

$res=odbc_exec($con, $sqlquery);
?>
```

Exploitation du résultat

```
bool odbc_fetch_row ( resource $result_id [, int $row_number ] )
```

Elle retourne :

- TRUE en cas de succès.
- FALSE s'il n'y avait plus de ligne, ou en cas d'erreur.

```
mixed odbc_result ( resource $result_id , mixed $field )
```

Elle retourne le contenu d'un champ.

field peut être aussi bien un entier, contenant le numéro de colonne du champ, dans le résultat, ou bien une chaîne de caractères, qui représente le nom du champ.

```php
<?php
$sqlquery="SELECT * FROM Employe;";
$res=odbc_exec($con, $sqlquery);
echo "<UL>";
while (odbc_fetch_row($res)) {
  $nom=odbc_result($res,"NomEmploye");
  $prenom=odbc_result($res,"PrenomEmploye");
  echo "<LI>$nom $prenom</LI>";
}
echo "</UL>";
odbc_close($con);
?>
```

SQLite (3)

SQLite

- SQLite est une bibliotheque écrite en C proposant un moteur de Base de données et implémentant en grande partie le standard SQL92.
- PHP inclut SQLite depuis sa version 5. Chaque base de données est stockée dans un fichier directement sur le disque où tourne PHP.
- Langage SQL un peu différent.
- `SQLite` est faiblement typé. Comme PHP, ce moteur accepte indifféremment du texte ou des nombres.
- On peut spécifier de manière indicative un type de données pour chaque champ lors de la création de la base.
- La suite concerne SQLite version 3. Il s'agit d'une API objet.

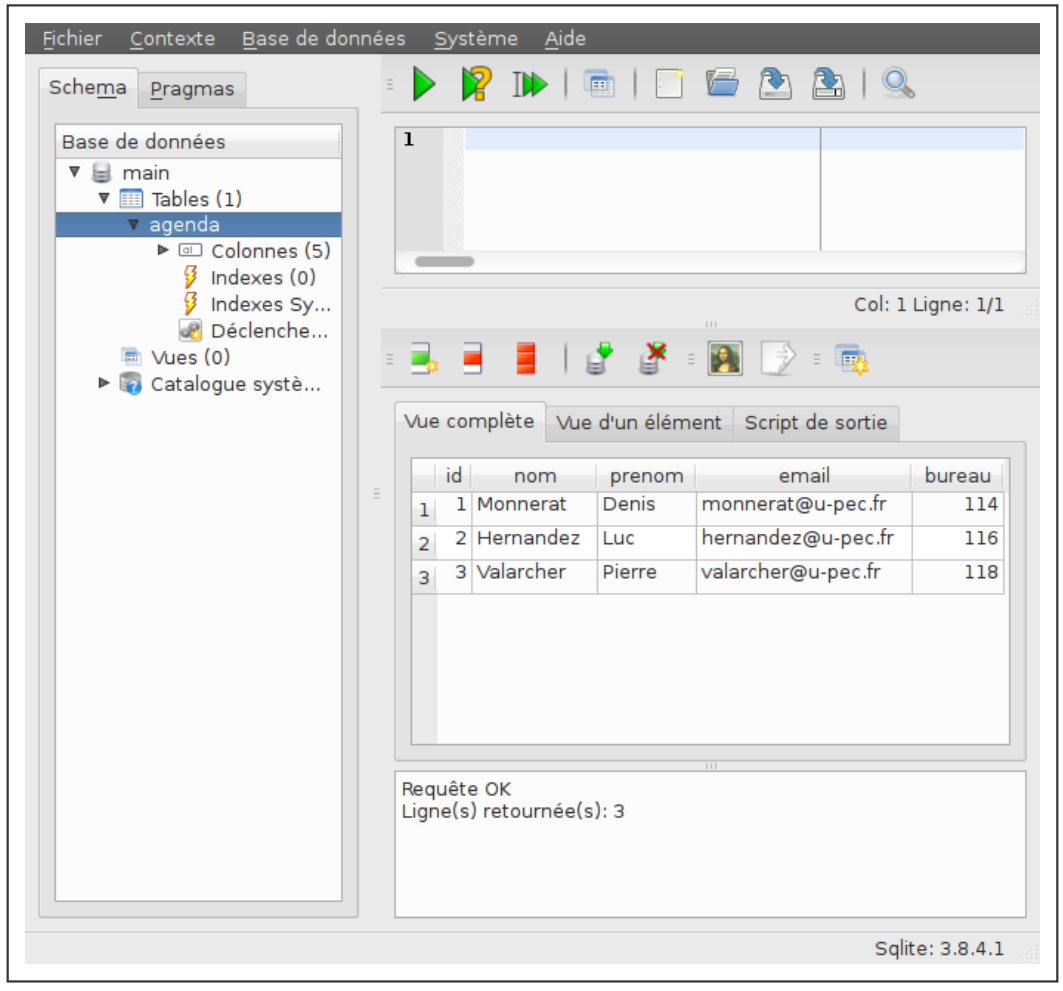
Il n'y a pas de types mais des "classes" de stockage.

```
CREATE TABLE "agenda" (
  "id" INTEGER PRIMARY KEY AUTOINCREMENT,
  "nom" TEXT,
  "prenom" TEXT,
  "email" TEXT,
  "bureau" INTEGER
);
```

NULL	La valeur est NULL
INTEGER	Entier signé signed sur 1, 2, 3, 4, 6, or 8 octets suivant la grandeur du nombre
REAL	Flottant sur 8 octets (IEEE floating point number)
TEXT	texte, stocké en utilisant l'encoage de la base (UTF-8, UTF-16BE ou UTF-16LE)
BLOB	données "binaires

documentation : `http://sqlite.org`

`http://sqliteman.com`, une application (écrite en C) de gestion de bases de données Sqlite3.



Api sqlite3

1. Connexion : ouverture d'un fichier.

```php
<?php
$db = new SQLite3('./agenda.db');
?>
```

2. Requêtes : Deux méthodes : `exec` et `query`.
   - `exec` pour les requêtes "sans résultats" (INSERT, UPDATE, ou DELETE).
   - `query` pour les requêtes de selections (SELECT).

```php
<?php
$query = "INSERT INTO users VALUES ( '$user', '$pass' )";
$db->exec($query) or die("Unable to add user $user");
?>
```

Requête query

```php
<?php
$db = new SQLite3('essai.db');
$results = $db->query('SELECT nom,prenom,email FROM agenda');
while ($row = $results->fetchArray()) {
  echo "<li>".$row['nom']." ".$row['prenom'].
    " : ".$row['email']."</li>";
}
?>
```

Requête préparée

```php
<?php
$db = new SQLite3('mysqlitedb.db');

$db->exec('CREATE TABLE foo (id INTEGER, bar STRING)');
$db->exec("INSERT INTO foo (id, bar)
  VALUES (1, 'Ceci est un test')");

$stmt = $db->prepare('SELECT bar FROM foo WHERE id=:id');
$stmt->bindValue(':id', 1, SQLITE3_INTEGER);

$result = $stmt->execute();
var_dump($result->fetchArray());
?>
```

PDO

PDO

- Interface commune d'accès à différents SGBD.
- Extension PHP qui est incluse dans la distribution standard depuis PHP 5.1
- Abstraction qui nécessite des drivers particuliers selon le sgbd.
- Possède trois classes :

`PDO`	la classe d'interaction principale avec le SGBD
`PDOStatement`	la classe gérant les résultats de requêtes, préparées, ou exécutées
`PDOException`	classe d'exception

Connexion

```php
<?php
$pdo = new PDO('mysql:host=localhost;dbname=tp3',
  'mylogin',
  'mypassword'
);
// iteration classique
$res = $pdo->query("SELECT nom FROM professeur");
while ($result = $res->fetch()) {
  echo $result['nom'];
}
// iteration avec l'interface traversable
foreach ($res as $result) {
  echo $result['nom'];
}


?>
```

Résultat

La méthode fetch de PDOStatement est contrôlée par le fetch_style, que l'on peut positionner en argument de cette méthode, ou avec la méthode `setFetchMode`

```php
<?php
$pdo = new PDO('mysql:host=localhost;dbname=tp3',
  'mylogin','mypassword');
$res = $pdo->query("SELECT nom FROM professeur");
$res->setFetchMode(PDO::FETCH_OBJ);

foreach ($res as $result) {
  echo $result->nom;
}
?>
```

Il existe d'autres façons de récupérer un jeu de résultat.

En particulier, PDO::FETCH_CLASS : retourne une nouvelle instance de la classe demandée, liant les colonnes du jeu de résultats aux noms des propriétés de la classe.

Requêtes

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

exécute une requête SQL dans un appel d'une seule fonction, retourne le nombre de lignes affectées par la requête.

PDO::exec() ne retourne pas de résultat pour une requête SELECT.

- Pour une requête SELECT dont vous auriez besoin une seule fois dans le programme, utilisez plutôt la fonction PDO::query().
- Pour une requête dont vous auriez besoin plusieurs fois, préparez un objet PDOStatement avec la fonction PDO::prepare() et exécutez la requête avec la fonction PDOStatement::execute().

Requêtes préparées

Emulées par PDO si le driver ne les supporte pas. Exemple en insert :

```php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value)
             VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();

// insertion d'une autre ligne
$name = 'two';
$value = 2;
$stmt->execute();
```

Remarques

- On peut utiliser ? comme marqueur de paramètre dans la requête :

```php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (?,?)");
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $value);

// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();
```

- La méthode `PDOStatement::bindValue` permet d'associer une valeur à un paramètre.
- On peut passer les paramètres de la requête dans la méthode `PDOStatement::execute` :

```php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (?,?)");

// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute(array($name,$value));
```

Requêtes préparées

Exemple en select

```php
<?php
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");
if ($stmt->execute(array($_GET['name']))) {
  while ($row = $stmt->fetch()) {
    print_r($row);
  }
}
?>
```

Transactions

- Atomicité, Consistance, Isolation et Durabilité (ACID).
- Toutes les bases de données ne supportent pas les transactions, donc, PDO doit s'exécuter en mode "autocommit" lorsque vous ouvrez pour la première fois la connexion.
- PDO::beginTransaction() pour l'initialiser.
- PDO::commit() ou la fonction PDO::rollBack() pour la terminer.

```php
/* Commence une transaction, désactivation de l'auto-commit */
$dbh->beginTransaction();

/* Insérer plusieurs enregistrements sur une base tout-ou-rien */
$sql = 'INSERT INTO fruit
  (name, colour, calories)
  VALUES (?, ?, ?)';

$sth = $dbh->prepare($sql);

foreach ($fruits as $fruit) {
  $sth->execute(array(
    $fruit->name,
    $fruit->colour,
    $fruit->calories
  ));
}

/* Valider les modifications */
$dbh->commit();

/* La connexion à la base de données est maintenant
 * de retour en mode auto-commit */
```

```php
<?php
/* Démarre une transaction, désactivation de l'auto-commit */
$dbh->beginTransaction();

/* Modification du schéma de la base ainsi que des données */
$sth = $dbh->exec("DROP TABLE fruit");
$sth = $dbh->exec("UPDATE dessert
  SET name = 'hamburger'");

/* On s'aperçoit d'une erreur et on annule les modifications */
$dbh->rollBack();

/* Le connexion à la base de données est maintenant de
 * retour en mode auto-commit */
?>
```