



Base de données / Type de bases de données

Objectifs PDO

PDO

Connexion à une BDD

Récupération des données

Ajouter des données

Editer des données

Supprimer des données

Démonstration Mini-Projet



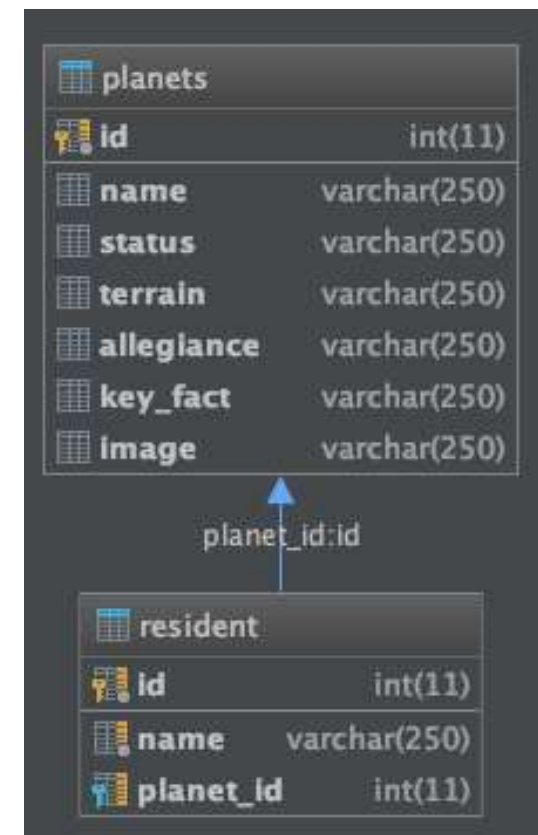


Base de données / Type de bases de données



Composition d'un schéma relationnel

- Une ou plusieurs tables
- Qui comprendront chacune une ou plusieurs colonnes
- Qui comprendront chacune 0 ou plusieurs contraintes (nullable, not null, unique, auto- increment)
- Nos tables seront en relation via l'utilisation de clé primaire et de clé étrangère



Pourquoi utiliser PDO

- Standardise la méthodologie de connexion aux systèmes de gestion de bases de données.
- Méthode la plus largement utilisée en PHP car elle simplifie les migrations de BDD.
- Avant on utilisait directement les méthodes mysql de PHP mais ces dernières sont dépréciées.
- Utilisation de la classe PDO directement dans PHP.
- Réception des données sous forme de tableau associatif, directement utilisable.
- Utilisé dans les frameworks PHP



La classe **PDO**, permet d'instancier des **connexions à une base de données** directement depuis **PHP**.

Elle contiendra de nombreuses **méthodes** utiles pour faire nos **requêtes SQL depuis nos fichiers PHP**.

Permettant aussi de **configurer** le comportement et les **données reçues** de notre base de données. (format de données reçu, gestion des erreurs mySql...)

La base de données renverra des **tableaux associatifs**.

Nous **combinerons** ces **tableaux** à nos **fonctions et boucles** afin de **créer des pages webs dynamiques** affichant des données réelles.

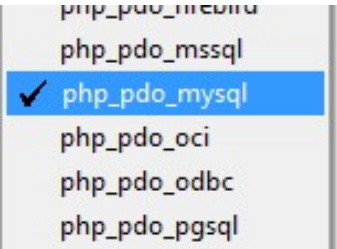
Et surtout à nos **formulaires** afin de permettre à des **utilisateurs** de **gérer les données** de manière **sécurisée** dans une **interface Web**.

(les administrateurs du site n'iront jamais taper des requêtes SQL ou consulter PHPMyAdmin)



Configurer notre environnement pour utiliser PDO

Normalement activé par défaut



L'extension suivante de votre php.ini ne dois pas être commenté (pas de « ; »). Par défaut aussi

extension=php_pdo_mysql.dll

```
www > phpinfo.php >
```

```
phpinfo();
```

PDO

PDO support	enabled	
PDO drivers	sqlite, mysql	

pdo_mysql

PDO Driver for MySQL	enabled	
Client API version	mysqlnd 5.0.12-dev - 20150407 - \$id: 38fea24f2b47fa75190d1be390c98ae0acafe387 \$	
Directive	Local Value	Master Value
pdo_mysql.default_socket	no value	no value

pdo_sqlite


PDO Driver for SQLite 3.x	enabled	
SQLite Library	3.15.1	

Création d'une Base de données avec une table pour l'exemple PDO

```
CREATE DATABASE garage12;
```

▷ Run

```
USE garage12;
```

▷ Run |  Select

```
CREATE TABLE car (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    model VARCHAR(255) NOT NULL,  
    brand VARCHAR(255) NOT NULL,  
    horsePower INT NOT NULL,  
    image VARCHAR(255) NOT NULL  
);
```

```
INSERT INTO car (model, brand, horsePower, image)  
VALUES ('Model S', 'Tesla', 670, 'modelS.jpg');
```

▷ Run

```
INSERT INTO car (model, brand, horsePower, image)  
VALUES ('Civic', 'Honda', 158, 'civic.jpg');
```

▷ Run

```
INSERT INTO car (model, brand, horsePower, image)  
VALUES ('Golf', 'Volkswagen', 150, 'golf.jpg');
```

▷ Run

```
INSERT INTO car (model, brand, horsePower, image)  
VALUES ('Mustang', 'Ford', 450, 'mustang.jpg');
```

▷ Run

```
INSERT INTO car (model, brand, horsePower, image)  
VALUES ('911 Carrera', 'Porsche', 379, 'carrera.jpg');
```

Connexion à la base de données

Création d'une instance PDO

```
$pdo = new PDO('mysql:host=localhost;dbname=test;charset=utf8', 'user', 'password');
```

Le constructeur de la classe PDO a besoin des paramètres suivants:

- mysql:host: - nom de domaine ou l'adresse IP de votre base de données mysql (localhost en local)
- dbname: nom de la base de données à utiliser
- charset: type d'encodage de votre BDD
- user: nom de l'utilisateur mysql que vous souhaitez utiliser
- password: mot de passe de l'utilisateur

Connexion PDO

Il est préférable de créer une fonction que nous utiliserons dans nos projets utilisant PDO

Voici un exemple simple

```
function connectDB() {  
  
    $host = 'localhost';  
    $dbName = 'marketplace';  
    $user = 'root';  
    $password = '';  
  
    return new PDO( dsn: 'mysql:host=.'.$host.';dbname=.'.$dbName.';charset=utf8', $user, $password);  
}
```

Il suffira de remplacer les variables dans la fonction et stocker la valeur de retour de la fonction
Pour créer une connexion à la BDD

Config PDO

Configuration PDO pour ajouter

- Les erreurs liées à la BDD
- Mode de récupération des datas

```
// Recevoir les erreurs PDO ( coté SQL )  
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
// Choisir les indices fetchs ASSOC, NUM, BOTH  
$pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
```

Utilisation d'un try/catch pour récupérer les erreurs

Si le code dans le TRY renvoie une erreur (Warning, Fatal Error...) celle-ci est envoyée dans le CATCH au lieu de s'afficher dans le résultat de la page

```
try {  
    $pdo = new PDO('mysql:host=' . $host . ';dbname=' . $dbName . ';charset=utf8',  
    $pdo->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);  
    $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE,PDO::FETCH_ASSOC);  
  
    return $pdo;  
} catch (Exception $e) {  
    echo("Erreur de connexion a la base de données. connectDB()");  
    die();  
}
```

Lorsque une erreur est « attrapée » celle-ci est accessible dans \$e

Il est également possible d'essayer une autre connexion etc..

Au lieu de simplement avoir une fatale error qui empêche le fonctionnement du code

Code final

Paramètres de connexion

Instanciation de la connexion

Configuration PDO

Return l'instance

Gestion erreur de connexion BDD

```
function connectDB(): PDO
{
    $host = 'localhost';
    $dbName = 'testDB';
    $user = 'root';
    $password = '';

    try {
        $pdo = new PDO(
            'mysql:host=' . $host . ';dbname=' . $dbName . ';charset=utf8',
            $user,
            $password
        );
        $pdo->setAttribute(
            PDO::ATTR_ERRMODE,
            PDO::ERRMODE_EXCEPTION
        );
        $pdo->setAttribute(
            PDO::ATTR_DEFAULT_FETCH_MODE,
            PDO::FETCH_ASSOC
        );
    }

    return $pdo;
} catch (Exception $e) {
    echo("Erreur de connexion a la base de données. connectDB()");
    die();
}
```

Code final

```
function connectDB(): PDO
{
    $host = 'localhost';
    $dbName = 'garage12';
    $user = 'root';
    $password = '';

    try {
        $pdo = new PDO(
            'mysql:host=' . $host . ';dbname=' . $dbName . ';charset=utf8',
            $user,
            $password
        );
        $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);

        return $pdo;
    } catch (Exception $e) {
        echo("Erreur de connexion a la base de données. connectDB()");
        die();
    }
}
```

Récupération de données

Faire une requête sans paramètre

Appel de la fonction query() de l'instance PDO

Elle nous permettra de faire des requêtes SQL sans paramètres

```
$pdo = connectDB();  
$requete = $pdo->query("SELECT * FROM car;");  
$cars = $requete->fetchAll();
```

La variable **\$response** contiendra un nouvel objet de type **PDOStatement**

La méthode **fetchAll()** récupère les **résultats** de la **requête** dans un **tableau PHP**

Que nous stockons dans une variable **\$cars**



fetchAll()

La méthode **fetchAll()** nous permettra de récupérer les données dans un tableau

Chaque ligne de la table sera un tableau associatif

Un indice associatif par champ

```
var_dump($reponse->fetchAll());
```

```
array (size=3)
  0 =>
    array (size=20)
      'id' => string '1' (length=1)
      0 => string '1' (length=1)
      'nom' => string 'Super restaurant' (length=16)
      1 => string 'Super restaurant' (length=16)
      'num_rue' => string '1' (length=1)
      2 => string '1' (length=1)
      'nom_rue' => string 'Champs-Elysee' (length=13)
      3 => string 'Champs-Elysee' (length=13)
      'ville' => string 'Paris' (length=5)
      4 => string 'Paris' (length=5)
      'code_postal' => string '75000' (length=5)
      5 => string '75000' (length=5)
      'type' => string 'Gastronomique' (length=13)
      6 => string 'Gastronomique' (length=13)
      'image_link' => string 'images/champ-elysee.jpeg' (length=24)
      7 => string 'images/champ-elysee.jpeg' (length=24)
      'star' => string '4' (length=1)
      8 => string '4' (length=1)
      'slug' => string 'gastro' (length=6)
      9 => string 'gastro' (length=6)
  1 =>
    array (size=20)
      'id' => string '2' (length=1)
      0 => string '2' (length=1)
      'nom' => string 'Kebab Clermont' (length=14)
      1 => string 'Kebab Clermont' (length=14)
      'num_rue' => string '10' (length=2)
      2 => string '10' (length=2)
      'nom_rue' => string 'Place de Jaude' (length=14)
      3 => string 'Place de Jaude' (length=14)
      'ville' => string 'Clermont-Fd' (length=11)
      4 => string 'Clermont-Fd' (length=11)
      'code_postal' => string '63000' (length=5)
      5 => string '63000' (length=5)
      'type' => string 'Fast Food' (length=9)
      6 => string 'Fast Food' (length=9)
```

fetchAll()

On remarque que les indices des tableaux internes sont doublés numérique / associatif

Il est possible de modifier la configuration a votre guise (associatif, numériques, les deux)

Nous avons déjà configuré cela avec le paramètre **PDO::FETCH_ASSOC** dans la configuration

```
array (size=3)
  0 =>
    array (size=20)
      'id' => string '1' (length=1)
      0 => string '1' (length=1)
      'nom' => string 'Super restaurant' (length=16)
      1 => string 'Super restaurant' (length=16)
      'num_rue' => string '1' (length=1)
      2 => string '1' (length=1)
      'nom_rue' => string 'Champs-Elysee' (length=13)
      3 => string 'Champs-Elysee' (length=13)
      'ville' => string 'Paris' (length=5)
      4 => string 'Paris' (length=5)
      'code_postal' => string '75000' (length=5)
      5 => string '75000' (length=5)
      'type' => string 'Gastronomique' (length=13)
      6 => string 'Gastronomique' (length=13)
      'image_link' => string 'images/champ-elysee.jpeg' (length=24)
      7 => string 'images/champ-elysee.jpeg' (length=24)
      'star' => string '4' (length=1)
      8 => string '4' (length=1)
      'slug' => string 'gastro' (length=6)
      9 => string 'gastro' (length=6)
```

fetch()

La méthode fetch()

La méthode fetch nous permettra de récupérer un seul enregistrement en BDD

Utile si vous attendez un seul résultat

```
var_dump($reponse->fetch());
```

```
array (size=20)
  'id' => string '1' (length=1)
  0 => string '1' (length=1)
  'nom' => string 'Super restaurant' (length=16)
  1 => string 'Super restaurant' (length=16)
  'num_rue' => string '1' (length=1)
  2 => string '1' (length=1)
  'nom_rue' => string 'Champs-Elysee' (length=13)
  3 => string 'Champs-Elysee' (length=13)
  'ville' => string 'Paris' (length=5)
  4 => string 'Paris' (length=5)
  'code_postal' => string '75000' (length=5)
  5 => string '75000' (length=5)
  'type' => string 'Gastronomique' (length=13)
  6 => string 'Gastronomique' (length=13)
  'image_link' => string 'images/champ-elysee.jpeg' (length=24)
  7 => string 'images/champ-elysee.jpeg' (length=24)
  'star' => string '4' (length=1)
  8 => string '4' (length=1)
  'slug' => string 'gastro' (length=6)
  9 => string 'gastro' (length=6)
```

Si aucun résultat n'est trouvé en BDD fetch() renvoie FALSE

Très utile lors de la récupération par ID et tester rapidement si une data est présente ou non

Requête préparée

Certaines requêtes ont besoin de paramètres, provenant souvent des utilisateurs.

Il ne faudra pas utiliser `query()` si votre requête a besoin de paramètres (`findBy` etc)

Préparer les requêtes permet de sécuriser les appels de notre base de données

Requêtes avec paramètres = obligation d'utiliser une requête préparée (sécurité contre injections)

Requête préparée

Utilisation des méthodes PDO `prepare()` et `execute()`

```
include("connectDB.php");

$pdo = connectDB();
$requete = $pdo->prepare("SELECT * FROM car WHERE id = :id;");
$requete->execute([
    'id' => $_GET['id']
]);
```

- Préparation de la requête en indiquant les paramètres avec :
- Exécution de la requête avec les paramètres du tableau les
- L'indice correspond au paramètre dans la requête ici `id`
- **Requêtes avec paramètres = obligation d'utiliser une requête préparée (sécurité contre injections)**

Exemple d'insertion de données

- 1) Validation des données en vérifiant la méthode POST, données valides...
- 2) Création d'un formulaire d'ajout HTML / PHP méthode POST

Ajout en BDD avec cette requête et les données reçues du formulaire

```
$requete = $pdo->prepare("INSERT INTO car(model, brand, horsePower, image)
VALUES(:model,:brand,:horsePower,:image);");
$requete->execute([
    'model' => $_POST['model'],
    'brand' => $_POST['brand'],
    'horsePower' => $_POST['horsePower'],
    'image' => $_POST['image'],
]);
```

Exemple Edition de données

- 1) Création d'un formulaire d'édition
- 2) Validation des données au-dessus du form en vérifiant la méthode POST, données valides...
- 3) Affichage du formulaire avec les valeurs pré-remplies récupérées en BDD
- 4) On effectue notre requête de mise à jour

```
$requete = $pdo->prepare("UPDATE car SET model = :model, brand = :brand, horsePower = :horsePower,  
image = :image WHERE id = :id;");  
// Exécuter la requête avec les valeurs envoyées par POST et l'ID de la voiture  
$requete->execute([  
    "model" => $_POST["model"],  
    "brand" => $_POST["brand"],  
    "horsePower" => $_POST["horsePower"],  
    "image" => $_POST["image"],  
    "id" => $car["id"]  
]);
```

Suppression de données

1. Page accessible via un lien contenant l'id de la ligne à supprimer
2. Vérification de l'existence de cet ID en BDD
3. Formulaire contenant un bouton de confirmation de suppression et un bouton pour annuler
4. Si la suppression est confirmée, suppression par ID

```
$requete = $pdo->prepare("DELETE FROM car WHERE id = :id;");  
  
$requete->execute([  
    "id" => $_GET["id"]  
]);
```