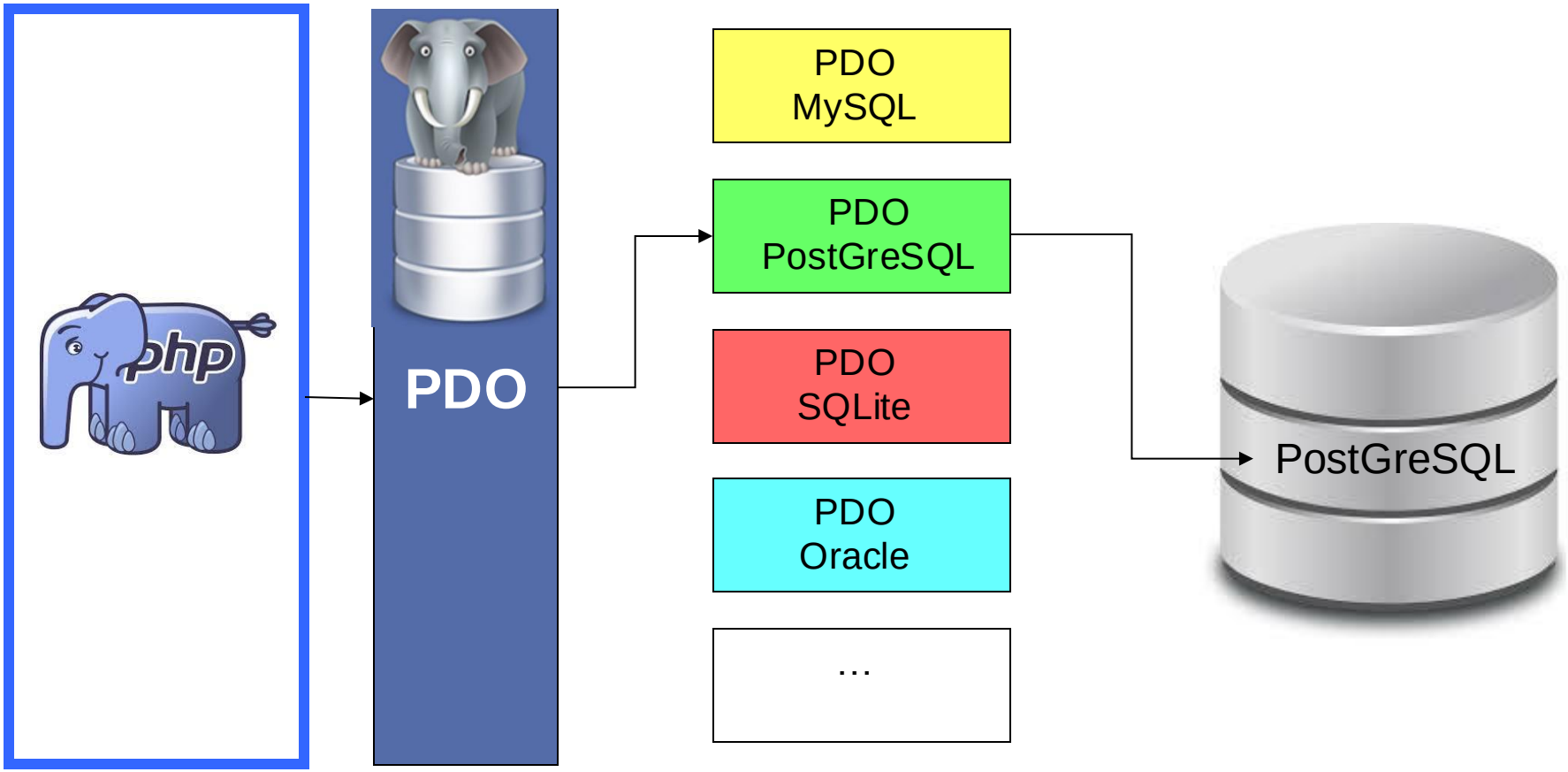
Chapitre n° 5 : Accès à une base de données

Exemple avec PDO et Sqlite

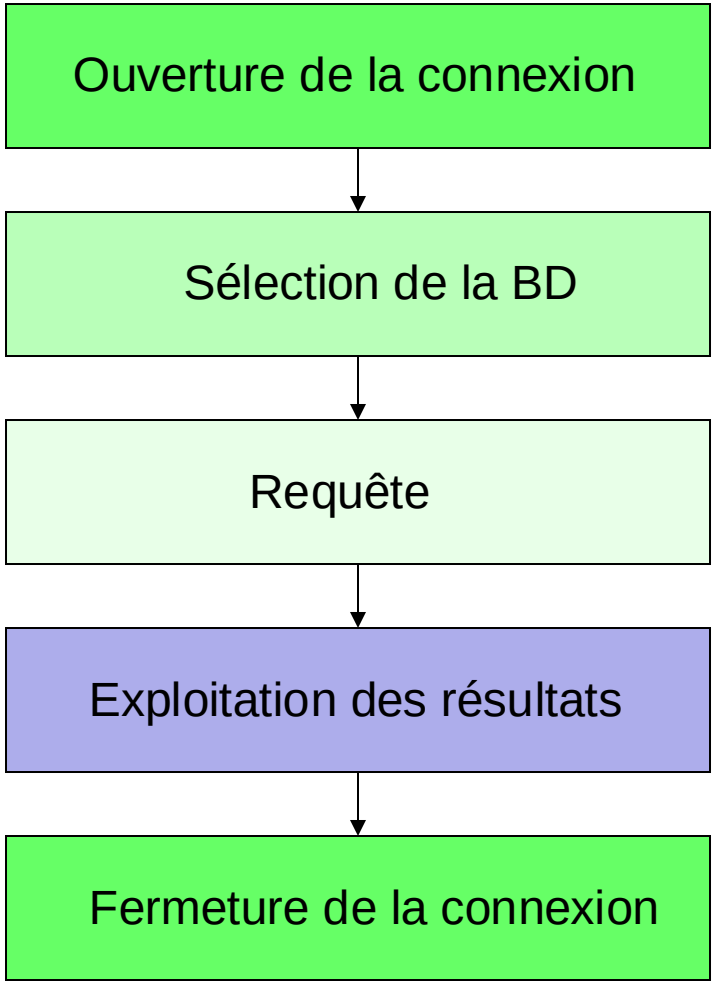
Accès aux Bases de Données

- Une des grandes forces de PHP => le support de nombreuses bases de données:
  - Sqlite, PostgreSQL, Ingres, Oracle, Sybase, IBM DB2, MySQL,…
  - PHP :
    - intègre un SGBDR, SQLite3
    - Interface PDO
- PDO (PHP Data Object) => interface pour accéder à une base de données depuis PHP
  - approche objet
  - socle commun pour les connecteurs vers les SGBD,
  - plus rapide que d'autres systèmes d'abstraction (PEAR DB, AdoDB,…)

Architecture des drivers PDO



Etapes d'accès à une base de données



3 classes principales

- Classe **PDO**: correspond au lien avec la BD
- Classe **PDOException**: permet le traitement des erreurs
- Classe **PDOStatement** : correspond aux requêtes et aux résultats

Ouverture de connexion

- Création d'une instance de la classe PDO:
  - Le paramètre du constructeur de la classe est le **DSN** : *Data Source Name*

```
$dbh = new PDO(DSN)
```

- DSN pour Sqlite

```
$dbh = new PDO('sqlite:path/newsDB','', '');
```

Simplement le chemin vers le fichier BD (pas l'URL !!)

- DSN pour MySQL

```
$dbh = new PDO('mysql:host=localhost;
  dbname=basetest', $user, $pass);
```

Fermeture de la connexion

- explicitement :
  - le destructeur de l'objet PDO ferme la BD
  - supprimer toutes des références à l'objet PDO

```
if ($dbh) {
    $dbh=NULL;
}
```

- implicitement :
  - à la fin du script PHP, tous les objets sont détruits.

Gestion des erreurs

- Gestion des erreurs de connexion

```
try {
   $dbh = new PDO('sqlite:data/fichier.bd);
}
catch (PDOException $e){
   die("erreur de connexion:".$e->getMessage());
}
```

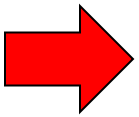
Requêtes & sécurité

IMPORTANT : injection de code dans PHP ou SQL
SQL : un autre langage en plus de PHP.

Risque : **fabriquer** du code pendant l'exécution, puis exécuter ce code.

Ex: $password = 'dX5#56';
$A='$'; $B='pass';$C='word=';$D='"";';
$CODE = $A+$B+$C+$D; eval($CODE);
=> modifie la variable ... password

Risque encore plus important si les données viennent de l'utilisateur ($_GET)



NE JAMAIS CONSTRUIRE DU CODE (PHP ou SQL) DANS UNE CHAINE

Requêtes réparées

- **Principe:**
  - Créer un modèle constant de requête : avec des zones variables
  - Compiler la requête (prépare) avec des valeurs (binding)
  - Exécution de la requête (execute)
  - Récupération des résultats (fetch)s
- **Avantages:**
  - Schéma de requête constante avec syntaxe correcte
  - Sécurité: éviter les attaques de type "injection SQL"
  - le SGBD sait ce qu'il doit recevoir; il vérifie les données transmises et fait les échappements nécessaires

Requêtes réparée

- Requête : dans une chaîne non modifiable

```
const SQL = 'SELECT nomv,pays FROM ville WHERE pays=:p';
```

zone variable **:p**

1. préparer (compiler) la requête :

```
$stm = $dbh->prepare(SQL);
```

2. Associer les zones variables avec des valeurs (binding)
3. Exécuter la requete : `$stm->execute();`
4. Récupérer les données dans un tableau de tableau

```
$table = $stm->fetchall();
```

Binding avec un 'array' dans execute()

- Construction du modèle de requête

```
const SQL = 'SELECT nomv,pays FROM ville WHERE pays=:p';
```

- Préparation du modèle de requête par le SGBD :

```
$stmt=$dbh->prepare(SQL);
```

- Binding et exécution

```
$nomv = $_GET['nom_ville'];
$stmt->execute([':nomv'=>$nomv]); // <= Binding de valeurs
```

- Récupération des résultats

```
$table = $stmt->fetchall();
```

Binding à une valeur séparé de l'exécution

- Construction du modèle de requête

```
const SQL="INSERT INTO ville (nomv,pays,descripville)
    values (:nomv, :pays, :descripville)";
```

- Préparation du modèle de requête par le SGBD :

```
$stmt=$dbh->prepare(SQL);
```

- Binding

```
$nomv="O'Sullivan";$pays='Irlande';$descripville="belle ville!";
$stmt->bindValue(':nomv',$nomv);
$stmt->bindValue(':pays',$pays);
$stmt->bindValue(':descripville',$descripville);
```

- Exécution

```
$stmt->execute();
```

- Récupération des résultats

```
$table = $stmt->fetchall();
```

Binding à une variable par référence

- bindParam : une variable par référence
- La valeur est récupérée dans la variable au moment de l'exécution
- Permet avec un seul binding de modifier les variables et lancer plusieurs exécutions successives
- Usage rare ... car ..
- Framework, une couche BD supplémentaire au dessus de PDO
- => connaitre juste le minimum de PDO

En pratique dans les TP : classe **DAO**

- Réalise l'ouverture de la BD
- Garantie une seule ouverture de BD (classe singleton)
- Masque la référence sur la BD (classe PDO)
- Une seule méthode : prepare()
  - retourne un objet de type PDOStatement
- Ne pas modifier
  - peut être considéré comme partie du framework

En pratique dans les TP : CRUD par classe

- Utiliser la classe DAO : `DAO::get()`
- CREATE : méthode d'instance, sauvegarde dans la BD
- READ : **méthode de classe**, lecture BD
- UPDATE : méthode d'instance, met à jour la BD
- DELETE : méthode d'instance, supprime dans la BD
  - attention : risque d'incohérence, contenu objet à détruire

- Ne pas modifier la BD sans passer par les méthodes, sinon risque incohérence BD et PHP

A retenir

- Accès à 99% des caractéristiques des BD par la couche **PDO**
- PDO utilise un modèle objet à 3 classes
  - la BD (PDO), le résultat (PDOStatement), une erreur
- Utilisation de drivers pour adapter PDO à une base de données particulière
- Ne faire que des requêtes préparées
- Binding de valeur (array) avec l'exécution (plus simple)
- Utiliser la classe DAO et les méthodes CRUD
- SQLITE : un BD simple dans un seul fichier
  - petite quantité de données ou pour la mise au point
  - SQLITE :penser à donner les **droits en écriture** sur le fichier **ET** le répertoire
  - préférer une BD avec un serveur : plus sur pour les données