PHP

PDO REQUETES PREPAREES

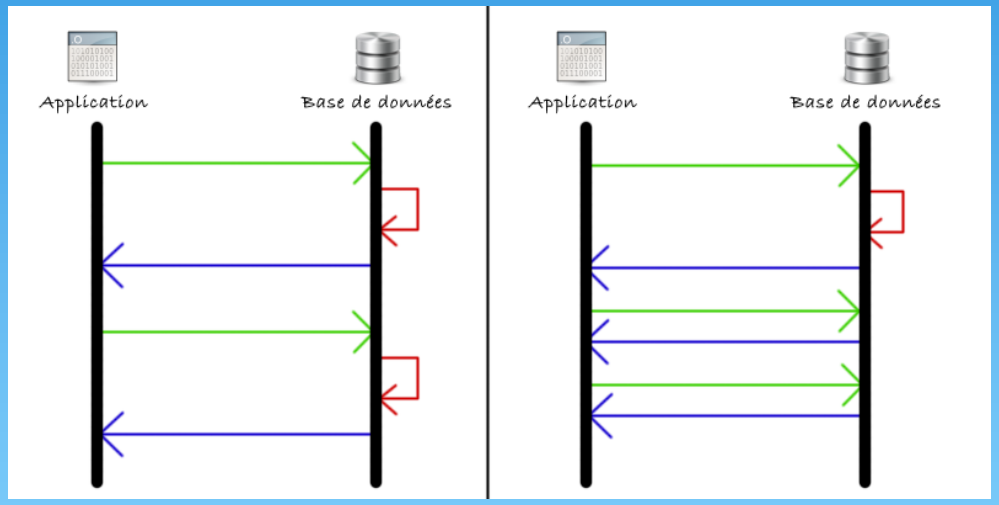
LIMITES A L'ACCES DIRECT

- **Performances**
  ré interprétation des requêtes à chaque exécution

- **Sécurité**
  données utilisateurs intégrées aux requêtes
  présence de caractères spéciaux (non quotés)
  → *injection SQL*

AVANTAGES DES REQUETES PREPAREES

- **Performance**



Légende : ***vert*** *: soumission de la même requête (même SELECT, WHERE qui change)*

rouge *: analyse*

bleu *: réponse de la base de données*

AVANTAGES DES REQUETES PREPAREES

- **Sécurité**

  injection SQL (insertion de code malveillant provenant de saisies non vérifiées)

- **Séparation du code (requête) et des données (paramètres)**
  injection SQL impossible

PRINCIPE

1. **Préparation de la requête**
   PDO::prepare()

2. **Liaison *(requête paramétrée)***
   PDOStatement::bindValue (),
   PDOStatement::bindParam();

3. **Exécution**
   PDOStatement::execute ()

4. **Traitement des tuples**
   PDOStatement::fetchXXX ()

prepare ()

- **Préparation de la requete**
  compilation et enregistrement sur la base de données
  renvoie un objet PDOStatement

```
<?php
// connexion à la base de données
$db =  new PDO('mysql:host=localhost; dbname=maBase', $login, $mdp);

$req = $db->prepare("SELECT nom, age  FROM PERSONNE order by nom");

…
?>
```

execute ()

- **Exécution de la requête**
  retour : *TRUE en cas de succès*
  FALSE en cas d'erreur

```
<?php
// connexion à la base de données
$db =  new PDO('mysql:host=localhost; dbname=maBase', $login, $mdp);

$req = $db->prepare("SELECT nom, age  FROM PERSONNE order by nom");

$req->execute ();
…
?>
```

TRAITEMENT DES TUPLES

- **Cf requêtes à accès direct**
  fetch(), fetchColumn(), fetchObject(), fetchAll(), …

```
<?php
// connexion à la base de données
$db =  new PDO('mysql:host=localhost; dbname=maBase', $login, $mdp);

$req = $db->prepare("SELECT nom, age  FROM PERSONNE order by nom");
$req->execute ();
while ($ligne= $req-> fetch(PDO::FETCH_ASSOC)){
                // $obj est un objet de la classe standard stdClass
                echo  $ligne["nom"]  . "  a " . $ligne["age"]. " ans.<br/>";
}

…
?>
```

bindColumn ()

- **Lier une colonne de la requête à une variable**
  retourne *TRUE en cas de succès*
  FALSE en cas d'erreur (colonne inexistante)
- **fetch()**
  PDO::FETCH_BOUND

```
<?php
// connexion à la base de données
$db =  new PDO('mysql:host=localhost; dbname=maBase', $login, $mdp);

$req = $db->prepare("SELECT nom, age  FROM PERSONNE order by nom");
$req->execute ();

$req->bindColumn ('nom', $nom);
$req->bindColumn ('age', $age);

while ($ligne= $req-> fetch(PDO::FETCH_BOUND)){
        echo  $nom . "  a " . $age. " ans.<br/>";
}
?>
```

REQUETE PARAMETREE (*v1*)

- **Intégration de paramètres dans la requête**
  syntaxe : **:parametre**

- **Attribution d'une valeur**
  lors de l'éxécution **PDOStatement::execute($valeursParam)**

- **Intérêt**
  quotation inutile
  réutilisation

REQUETE PARAMETREE (*v1*)

- **Exemple**

```
<?php
// connexion à la base de données
$db =  new PDO('mysql:host=localhost; dbname=maBase', $login, $mdp);

$req = $db->prepare("SELECT nom, age  FROM PERSONNE  WHERE ID=:IDPersonne ");

// tableau des valeurs de parametres
$valeursParam = array(":IDPersonne" => 17);
$req->execute ($ valeursParam );

if ($ligne = $req-> fetch(PDO::FETCH_ASSOC) {
                …
}
?>
```

bindValue ()

- **Lie un paramètre avec une valeur**

```
<?php
// connexion à la base de données
$db =  new PDO('mysql:host=localhost; dbname=maBase', $login, $mdp);

$req = $db->prepare("SELECT nom, age  FROM PERSONNE  WHERE ID=:IDPersonne ");

// attribution d'une valeur au paramètre IDPersonne
$req->bindValue(":IDPersonne", 17);
$req->execute ();

if ($ligne = $req-> fetch(PDO::FETCH_ASSOC) {
                    …
}
?>
```

bindParam ()

- **Lie un paramètre avec une variable**

  changer la valeur de la variable pour ré exécuter la requête

```php
<?php
$req = $db->prepare("INSERT INTO PERSONNE  (nom, age) VALUES (:nom, :age)");
$req->bindParam(':nom', $nom);
$req->bindParam(':age', $age);

// insertion d'une ligne
$nom = 'DUPONT';
$age = 8;
$req->execute();

// insertion d'une autre ligne avec des valeurs différentes
$nom = 'MARTIN';
$age = 12;
$req->execute();
?>
```

OUTILS

- **Nombres de lignes affectées**
  PDO:rowCount()

- **Identifiant de la dernière ligne insérée**
  PDO::lastInsertedId ()

- **Transactions**
  PDO::beginTransaction()
  PDO::commit ()
  PDO::rollback()

rowCount()

- **Nombre de lignes affectées par la dernière requête**
  uniquement pour INSERT, UPDATE, DELETE
  après l'exécution de la requête (**execute()**)

- **Cas d'un SELECT**
  dépend de la base de données (retour non garanti)
  solution : compter avant : **SELECT COUNT(*) ….**

```
<?php
// suppression des pistes de l'album n° 17
$req = $db->prepare('DELETE FROM Piste where IDAlbum = 17');
$req->execute();

/* Retourne le nombre de pistes effacées */
$total = $req->rowCount();
print("Effacement des $total  pistes de l'album.");
?>
```

lastInsertID ()

- **Identifiant de la dernière ligne insérée**

  Cas de clé auto-incrémentée (utilisable dans une requête ultérieure)
  Rq : spécifique à la connexion (i.e. « thread safe »)

```
<?php
// insertion du nouvel album
$reqAlbum = $db->prepare('INSERT INTO Album (titre)  VALUES (:titre)');
$reqAlbum->bindValue (':titre', 'The Joshua Tree');
$reqAlbum->execute();
$IDAlbum = $db->lastInsertID ();

// insertion des pistes de l'album
$reqPiste = $db->prepare('INSERT INTO Piste (numero, titre, duree, IDAlbum )
                          VALUES (:numero, :titre, :duree, :IDAlbum)');
$valeursParam = array (  ':numero' => 1 ,
                         ':titre' => 'Where the Streets Have No Name',
                         ':duree' => '5:36',
                         ':IDAlbum' => $IDAlbum);
$req->execute($valeursParam);
?>
```

TRANSACTIONS

- **Contexte**
  *exécution groupée d'un lot de requêtes (**transaction**)*
  échec de l'une des requête annule tout le lot

- **Principe**
  1- marquage du début du lot **PDO::beginTransaction ()**
  2- exécution de requêtes (SELECT, INSERT, UPDATE, DELETE)
  3- validation du lot **PDO::commit()**
  ou annulation **PDO::rollBack()**

beginTransaction()

- **Débute une transaction**
  uniquement sur des tables de type InnoBD (pas MyISAM)

commit ()

- **Valide une transaction**
  exécution de toutes les requêtes entre le begin et le commit

rollBack()

- **Annule une transaction**
  toutes les requêtes entre le begin et le rollBack

TRANSACTION - exemple

- **Enregistrement d'un album**

```
<?php
 try {
                $db->beginTransaction();

                // ici les requêtes SQL
                // insertion de l'album
                …
                // insertion des pistes de l'album
                …
                $db->commit();
} catch (PDOException $e) {
                …
                $db->rollBack();
}
?>
```

CONCLUSION