Les requêtes préparées

Sécuriser les interactions avec la BDD

Pourquoi ?

Lorsque l'on effectue une requête SQL dans un fichier PHP, il peut arriver que la requête ne soit pas dynamique, c'est-à-dire qu'elle sera toujours la même. Par exemple :

```php
$requete = "SELECT * FROM users";
```

Mais si on commence à interagir avec des formulaires, avec $_GET, ou avec d'autres données **dynamiques**, alors la requête peut devenir dangereuse. Par exemple si on a des données venant du tableau **$_GET** dans la requête, alors il va être possible pour un utilisateur malintentionné de mettre dans l'URL des instructions qui vont se retrouver dans ma requête.

Exemple de requête non sécurisée :

```php
$requete = "SELECT * FROM users WHERE id = " . $_GET['id_user'];
```

Les injections SQL

On appelle « Injection SQL » le fait d'utiliser une faille de sécurité dans une requête pour exécuter du SQL contre le gré du développeur. Par exemple ici, on pourrait passer dans l'URL **id_user=1; DROP TABLE users;**

Si on laisse faire, cela peut permettre aux utilisateurs malintentionnés de manipuler notre base de données à notre insu et de causer des dommages potentiellement irréversibles, ou alors de se connecter en tant qu'admin en rajoutant une condition dans WHERE qui se vérifie tout le temps comme **OR 1 = 1** etc…

Une requête d'authentification pourrait alors devenir :

```php
$requete = "SELECT * FROM users WHERE login = admin AND password = 1 OR 1 = 1";
```

Avec une requête comme ça, même plus besoin de taper un mot de passe et vous pouvez vous connecter comme administrateur.

Les failles XSS (Cross-site scripting)

De la même façon, si on accepte dans notre base de données les contenus d'un formulaire, on peut se retrouver avec des balises <script> exécutant du code malveillant dans notre base. Dès que le contenu de cette base sera affiché, le script sera exécuté de manière invisible, et peut alors causer des dommages important comme de la collection de données, de la collection d'informations en session, de l'interception de formulaires contenant des mots de passe etc…

Comment s'en prémunir ?

Face à ces problématiques particulièrement dangereuses, il existe heureusement une solution simple : préparer ses requêtes.

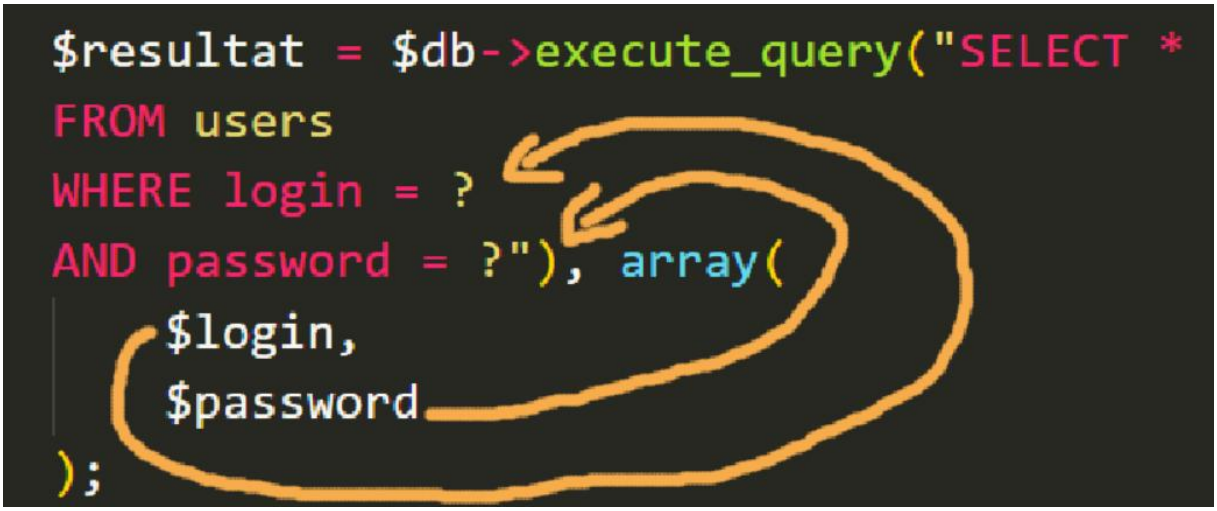
On appelle une requête préparée une requête dans laquelle on ne met pas les données dynamiques de nous-même, mais on va laisser un outil sécurisé **construire notre requête** à notre place.

Nous allons alors simplement prévoir les emplacements de ces données dans notre requête, puis lister les données en questions, et l'outil va s'occuper de vérifier le contenu de ces données avant d'exécuter la requête.

Avec la classe mysqli, on peut préparer nos requêtes de deux façons différentes, soit avec **execute_query**, soit avec **prepare**. Les deux présentent des avantages et des inconvénients, étudions ça plus en profondeur :

->execute_query()

Probablement la méthode la plus simple, puisqu'elle s'exécute en une fois avec deux paramètres bien précis : d'abord la requête contenant les **emplacements** de mes paramètres, symbolisés par des **?**, un peu comme un texte à trous, et ensuite un **tableau** contenant tous les **paramètres** à placer dans l'ordre dans leurs emplacements assignés. On retrouve alors la syntaxe suivante :



La variable **$login** va aller dans le premier emplacement prévu, et la variable **$password** dans le deuxième.

La variable **$resultat** va devenir un objet de la classe **mysqli_result**, contenant notamment la propriété **num_rows** qui peut nous permettre de vérifier le nombre de résultat retournés par la requête. Ce résultat est un objet que l'on appelle **Iterable**, ce qui veut dire que l'on n'a pas besoin de le transformer en tableau pour l'utiliser dans un **foreach**.

Si toutefois on a vraiment besoin d'un tableau, on peut utiliser **fetch_assoc()** pour obtenir seulement le premier résultat dans un tableau, ou alors **fetch_all(MYSQLI_ASSOC)** pour tout récupérer dans un tableau à deux dimensions.

Attention : Selon la requête utilisée, la variable $resultat n'aura pas forcément le même contenu. Si on fait une requête de sélection, on aura bien un résultat Iterable que l'on peut utiliser dans un foreach, mais si on fait un INSERT, une suppression etc… On aura un booléen permettant de tester si l'insertion s'est bien passée. Dans ce cas précis, on pourra alors accéder à la propriété affected_rows mais pas sur $resultat, sur l'objet $db directement.

->prepare()

Méthode légèrement plus complexe puisqu'elle induit plusieurs étapes et un déroulé peut-être moins intuitif, mais en revanche elle propose une sécurité supplémentaire non négligeable : la vérification du **typage** des données passées à la requête.

La logique de départ est presque la même : on prépare notre requête avec un texte à trous, où les **emplacements** sont symbolisés par des **?**

```
$email = "bastien.leroy@example.com";
$stmt = $db->prepare("SELECT *
FROM utilisateurs
WHERE email = ?");
```

Ensuite, on va **lier** les paramètres à leurs emplacements avec la méthode **bind_param**, mais contrairement à **execute_query()** on va en plus préciser le **type** de chaque variable, dans l'ordre, avec **s** pour les chaînes de caractère, **i** pour les nombres entiers, **d** pour les décimaux :

```
$stmt->bind_param("s", $email);
```

Et enfin, on va dire à notre variable **$stmt**, qui a maintenant toutes les informations dont elle a besoin, d'exécuter la requête :

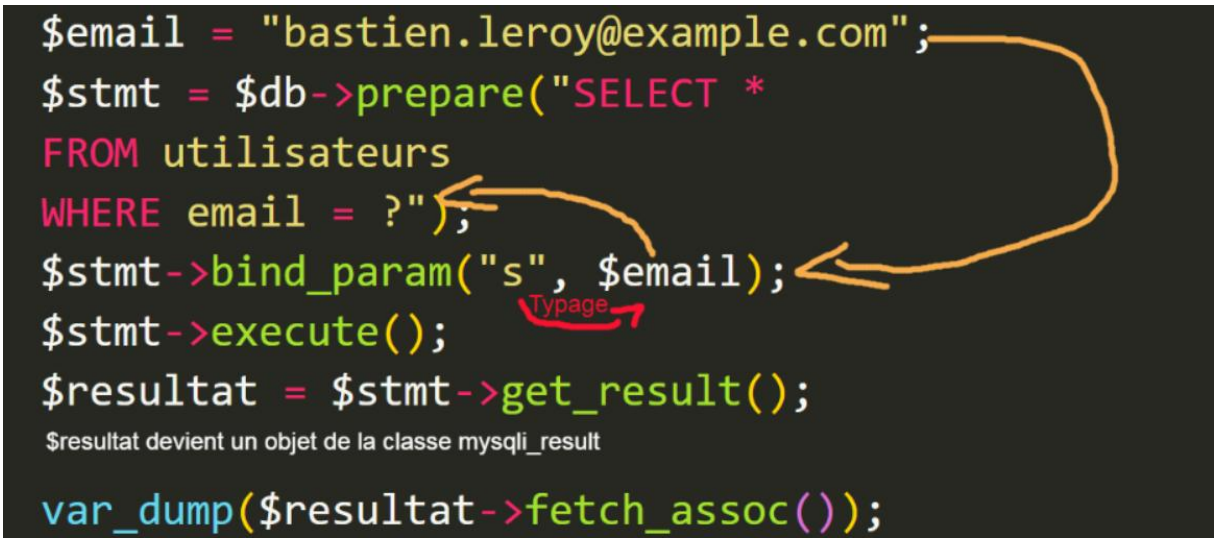
```
$boolean_resultat = $stmt->execute();
```

Comme indiqué ici, **execute()** va nous renvoyer un **booléen**, vrai ou faux, pour savoir si la requête s'est bien exécutée. Pour récupérer le résultat, on va devoir utiliser la méthode **get_result() sur $stmt** comme ceci :

```
$resultat = $stmt->get_result();
```

Et là on retombe sur nos pieds avec des nouveau un objet **mysqli_result**, comme ce que l'on obtient avec la méthode **execute_query()**. À noter cependant que si l'on fait des requêtes d'insertion ou de suppression etc, alors on a accès directement à la propriété **affected_rows** sur **$stmt**, tout comme la propriété **insert_id** en cas d'insertion.

Syntaxe complète avec prepare() :



Par où commencer ?

Afin de vous aider, je vous recommande de suivre la méthode suivante lorsqu'une consigne vous est demandée :

I. Rédigez la requête si possible dans PHPMyAdmin ou DBeaver. Faites fonctionner cette requête en mettant dans un premier temps des paramètres fictifs, ou alors ceux dont vous avez besoin.
II. Une fois votre requête rédigée et fonctionnelle, identifiez quels sont les éléments qui vont devenir **dynamiques** dans votre requête. Ces éléments vont devenir vos **emplacements « ? »**.
III. Votre requête va avoir besoin de **paramètres** à placer dans ces **emplacements**. Regardez donc si besoin dans votre fichier PHP comment obtenir ces paramètres : les récupérer dans des variables, dans la signature d'une fonction, dans $_POST ou $_GET… **var_dump est votre ami**.
IV. Maintenant que vous avez préparé votre texte à trous, et que vous connaissez tous vos **mots à placer**, importez votre requête dans votre fichier PHP. Choisissez **prepare()** ou **execute_query()** selon votre préférence.
V. Liez les paramètres à votre requête dans le même ordre que les emplacements.
VI. Exécutez la requête et vérifiez son résultat, soit avec un var_dump(), soit en transformant le résultat en **tableau** avec **fetch_assoc()** ou **fetch_all(MYSQLI_ASSOC)**
VII. Profitez.