

PDO et accès à la BD

PDO

PDO est l'acronyme pour PHP Data Object. C'est une interface pour accéder à plusieurs types de bases de données. L'utilisation de cette interface permet de changer la base de données avec un minimum d'impact dans le code.

PDO VS MySQLi

PDO

MySQLi

PDO VS MySQLi

PDO



► Autres...


MySQLi



Connexion avec PDO

Ligne DSN : Data source name. Il est nécessaire d'y définir l'encodage ici.

```
$pdo = new PDO(  
    'mysql:host=localhost;  
    dbname=nom_bd;  
    charset=utf8mb4',  
    $username,  
    $password,  
    $options  
);
```



Connexion PDO

Une stratégie doit être créée pour éviter que plusieurs PDO soient créés. Il est conseillé de réutiliser le même. Le singleton représente une bonne stratégie.

Options


Il est possible de passer des options à l'aide d'un objet de la forme Hashmap (« tableau associatif »)

Options de connexion

```
$options = array(  
    PDO::ATTR_ERRMODE=>  
        PDO::ERRMODE_EXCEPTION  
);
```



```
$options = [  
    PDO::ATTR_ERRMODE=>  
        PDO::ERRMODE_EXCEPTION  
];
```



Au lieu d'écho des warning lors d'erreurs, il y aura le lancement d'une exception qui pourra être traitée avec un try catch

Gestion des erreurs de connexion

```
try {  
    $pdo = new PDO(...);  
    echo "Connected\n";  
} catch (Exception $e) {  
    die("Unable to connect: " . $e->getMessage());  
}
```

Requête avec PDO

```
$reponse =  
    $pdo->query(  
        'SELECT * FROM tbl_entreprise'  
    );
```

Récupération de données avec PDO

```
$sql_query = 'SELECT * FROM tbl_user;'  
foreach ($pdo->query($sql_query) as $row)  
{  
    print $row['first_name'] . "\t";  
    print $row['last_name'] . "\t";  
}
```

Récupération directement dans une classe

```
$users = $pdo->query(  
    'SELECT * FROM tbl_user'  
)->fetchAll(  
    PDO::FETCH_CLASS, 'UserDTO'  
);
```

```
$users = $pdo->query(  
    'SELECT * FROM tbl_user'  
)->fetchAll(  
    PDO::FETCH_CLASS, 'UserDTO'  
);
```

Prepared statement

Le prepared statement permet de s'assurer que nous ne serons pas vulnérables à une faille du type « SQL injection ».

Injection SQL

```
SELECT
    user_name
FROM
    tbl_user
WHERE
    id_user = $valeur_formulaire;
```


Entrée normale

```
SELECT  
    user_name  
FROM  
    tbl_user  
WHERE  
    id_user = 1;
```

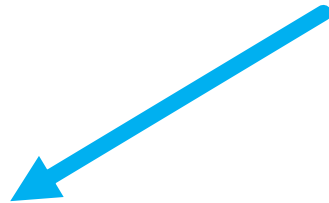
\$valeur_formulaire = « 1 »



Entrée normale

```
SELECT
    user_name
FROM
    tbl_user
WHERE
    id_user = 1;
```

```
SELECT
    user_name
FROM
    tbl_user;
```

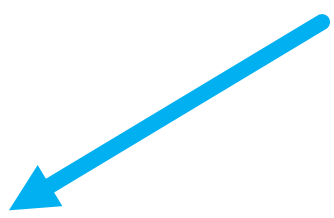


\$valeur_formulaire = « 1 ;Select
user_name, password from
tbl_user »

Entrée normale

```
SELECT
    user_name
FROM
    tbl_user
WHERE
    id_user = 1;
INSERT INTO tbl_admin(
    admin_name, password
) VALUES (
    'hacker', 'password'
);
```

\$valeur_formulaire = « 1 ;INSERT
INTO tbl_admin(admin_name,
password) VALUES ('hacker',
'password') »



Prepared statement

```
$statementHandle = $pdo->prepare('
```

```
SELECT
```

```
    user_name
```


```
FROM
```

```
    tbl_user
```

```
WHERE
```

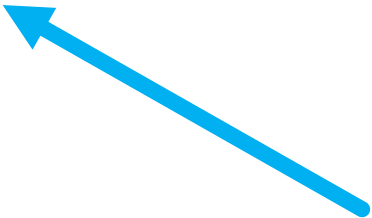
```
    id_user = :id_user');
```

Le symbole deux points indique qu'un paramètre doit être fourni à cette requête. Le nom du paramètre dans ce cas est id_user;



Prepared statement

```
$statementHandle->execute([  
    'id_user'=>$valeur  
]);
```



On exécute la requête avec le paramètre qui prendra la valeur de la variable \$valeur. Cette valeur est automatiquement échappée(escape) pour être sécuritaire aux injections SQL.