

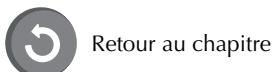
POO avancée : le CRUD

Sommaire

I. La méthode de lecture ou récupération des brèves : <i>getBreve</i>	3
II. La méthode de suppression : <i>deleteBreve</i>	4
III. Testons la classe	5

Crédits des illustrations:
© Fotolia, DR.

Les repères de lecture



Retour au chapitre



Définition



Objectif(s)



Espace Élèves



Vidéo /
Audio



Point important /
À retenir



Remarque(s)



Pour aller
plus loin



Normes et lois



Quiz

Très classiquement, les besoins liés à la base de données seront l'insertion (ou création), la lecture (récupération pour affichage par exemple), la modification et la suppression des données. L'acronyme **CRUD** est un bon moyen mnémotechnique : C pour Create, R pour Read, U pour Update et D pour Delete.

Nous allons donc intégrer les opérations CRUD dans la classe gestionnaire breveManager.

Ayant réalisé l'insertion des données, penchons-nous sur les opérations R, U et D.

```

gl
$token
{
    if ($id) {
        $ch ($id) {
            case T_COMMENT:
            case T_ML_COMMENT:
            case T_DOC_COMMENT:
                break;
            default:
                $ret = $text;
                break;
        }
    }
    return trim(str_replace(array('<','>'),array(
        if (!defined(T_ML_COMMENT))
        else define('T_DOC_COMMENT', T_ML_COMMENT);
        function strip_comments($source) {
            $ret = "";
            foreach ($tokens as $token) {
                if (is_string($token)) {
                    $ret .= $token;
                } else {
                    list($id, $text) = $token;
                    switch ($id) {
                        case T_COMMENT:
                        case T_ML_COMMENT: // we've
                        case T_DOC_COMMENT: // and
                            break;
                    }
                }
            }
        }
    ));
}

```

Fig.1

I. La méthode de lecture ou récupération des brèves : *getBreve*



Cette méthode peut prendre en argument soit une valeur vide, afin que toutes les brèves soient restituées, soit une valeur numérique dans le cas où nous souhaitons récupérer une seule brève, et cette valeur numérique sera naturellement celle correspondant à l'identifiant de la brève.

```

public function getBreve ($id = '')
{
    if(empty($id)) // toutes les brèves
    {
        $sql = 'SELECT id, text, dt_creation FROM breve';
        $stmt=$this->db->prepare($sql);
    }
    elseif (is_numeric($id)) // une brève, identifiée par son ID
    {
        $sql = 'SELECT id, text, dt_creation FROM breve WHERE
id =:id';
        $stmt=$this->db->prepare($sql);
        $stmt->bindParam(':id', $id);
    }
    $stmt->execute();
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC))
    {
        $result[] = $row;
    }
    return $result;
}

```

Fig.2

À noter l'utilisation de la commande `FETCH_ASSOC` qui permet d'éliminer les index numériques retournés par MySQL et de ne conserver que les index associatifs (issus des intitulés des colonnes de la table).

Avec `FETCH_ASSOC`:

```
0 =>
    array
        'id' => string '11' (length=2)
        'text' => string 'texte' (length=10)
        'dt_creation' => string '2013-11-08' (length=10)
```

Fig.3

Sans `FETCH_ASSOC`:

```
array
0 =>
    array
        'id' => string '11' (length=2)
        0 => string '11' (length=2)
        'text' => string 'texte' (length=10)
        1 => string 'texte' (length=10)
        'dt_creation' => string '2013-11-08' (length=10)
        2 => string '2013-11-08' (length=10)
```

Fig.4

La méthode de mise à jour nommée `updateBreve`.

Elle attend en argument un objet et ne pose pas de difficultés particulières.

```
public function updateBreve (Breve $breve) // objet Breve attendu
{
    $sql = 'UPDATE breve SET text = :text, dt_creation = :dt_creation
            WHERE id = :id';
    $stmt=$this->_db->prepare($sql);
    $stmt->bindParam(':id', $breve->getId());
    $stmt->bindParam(':text', $breve->getText());
    $stmt->bindParam(':dt_creation', $breve->getDt_creation());
    $stmt->execute();
}
/*
```

Fig.5

II. La méthode de suppression: `deleteBreve`

Elle attendra comme argument une valeur numérique et non pas un objet. En effet, comme nous l'avons vu avec l'exemple de l'insertion, l'utilisation d'un objet afin d'utiliser une méthode implique de fait de créer cet objet au préalable. Ceci peut se révéler être une contrainte dans un développement sans que cela n'apporte d'intérêt relatif à la vérification ou le contrôle des données car un identifiant n'est qu'une simple valeur numérique.

Par ailleurs, si dans un espace d'administration nous avons une liste des brèves sous forme de liens destinés à être supprimées, il n'est sans doute pas pertinent de recréer un objet à chaque itération (tour de boucle) de la liste.

```

public function deleteBreve ($id)
{
    $sql = 'DELETE FROM breve WHERE id =:id';
    $stmt=$this->_db->prepare($sql);
    $stmt->bindParam(':id', $id);
    $stmt->execute();
    $count = $stmt->rowCount();
    return $count;
}

```

Fig.6

Nous utilisons ici `rowCount` qui est une méthode très pratique de la classe PDO permettant d'obtenir le nombre de lignes impactées par la requête. Dans le cas de notre requête de suppression, si celle-ci est réussie, une seule ligne est impactée (une suppression demandée). Il suffit, lors de l'appel de cette requête, de récupérer la valeur dans une variable pour l'utiliser, par exemple :

```

$deleteBreve = $manager->deleteBreve(1);
echo $deleteBreve;

```

Fig.7

Ceci affiche 1 ou 0 si aucune ligne n'est impactée (par exemple la brève a déjà été supprimée et n'existe plus dans la base).



Page du manuel relative à `rowCount`:

<http://php.net/manual/en/pdostatement.rowcount.php>

III. Testons la classe

Il ne nous reste plus qu'à tester le bon fonctionnement des deux classes en commençant par l'insertion d'un nouveau contenu.

Préparation d'un contenu test:

```

$breve_data = array('id' => 1, 'text' => 'Texte pour le contenu',
'dt_creation' => '2013-10-14' );

```

Fig.8

Instanciation de la classe `Breve`. Nous créons ainsi un objet `breve` avec le contenu créé précédemment:

```

$breve = new Breve($breve_data);

```

Fig.9

Instanciation de PDO permettant de créer la connexion à la base de données. Nous affectons ce résultat à la variable `$db`:

```

$db = new PDO('mysql:host=localhost;dbname=emweb_news', 'root',
 '');

```

Fig.10

Instanciation de la classe *breveManager* en passant au constructeur la connexion à la base de données créée précédemment:

```
$manager = new breveManager($db);
```

Fig.11

Pour finir, appel de la méthode d'ajout d'une brève dans la base:

```
$manager->addBreve ($breve);
```

Fig.12

Vérification du résultat dans MySQL:

```
select * from breve;
```

id	text	dt_creation
1	Texte pour le contenu	2013-10-14

Fig.13

Testons la suppression d'une brève en appelant la méthode de suppression:

```
$manager->deleteBreve (1);
```

Fig.14

Testons l'affichage de toutes les brèves en utilisant la méthode *getBreve()* sans utiliser d'arguments:

```
$breves = $manager->getBreve();  
var_dump ($breves);
```

Fig.15

```
array  
0 =>  
    array  
        'id' => string '1' (length=1)  
        'text' => string 'Texte pour le contenu' (length=21)  
        'dt_creation' => string '2013-10-14' (length=10)  
1 =>  
    array  
        'id' => string '2' (length=1)  
        'text' => string 'Texte pour le contenu' (length=21)  
        'dt_creation' => string '2013-10-14' (length=10)
```

Fig.16

Testons maintenant l'affichage d'une brève définie en passant cette fois à la méthode `getBreve` l'identifiant souhaité, par exemple 2 :

```
// attention à ne pas nommer la variable avec le même nom que  
l'objet breve  
$current_breve = $manager->getBreve(2);  
var_dump($current_breve);
```



```
array  
0 =>  
array  
'id' => string '2' (length=1)  
'text' => string 'Texte pour le contenu' (length=21)  
'dt_creation' => string '2013-10-14' (length=10)
```

Fig.17

Et pour terminer, testons la mise à jour d'une brève avec la méthode `updateBreve`. Nous avons obligé cette méthode à ne fonctionner que si un objet lui était passé en paramètre. Il nous faut donc en premier lieu créer l'objet avec les valeurs souhaitées puis utiliser cet objet pour effectuer la mise à jour.

Supposons que nous avons plusieurs brèves dans la base de données et que nous souhaitons mettre à jour celle dont la valeur de l'identifiant est 2. Décomposons les étapes dont les objectifs et l'enchaînement nécessitent d'être bien compris.

Nous récupérons donc depuis la base de données les informations de la brève avec la méthode `getBreve(2)` comme vu précédemment :

```
$current_breve = $manager->getBreve(2);
```

Fig.18

Puisque nous allons instancier la classe `Breve` nous préparons le tableau de données qui servira au constructeur à construire l'objet :

```
$new_content = array('id' => (int) $current_breve[0]['id'], 'text'  
=> $current_breve[0]['text'], 'dt_creation' => $current_breve[0]  
['dt_creation'], );
```

Fig.19

Ici un point important est que la valeur de l'identifiant `id` doit être un entier numérique(`integer`) car nous avons effectué cette vérification avec `is_int` dans la méthode `setId`.

```
if((is_int($id)) AND ($id > 0)) {  
    $this->_id = $id;  
}
```

Fig.20

Or si nous effectuons un `var_dump` nous constatons que le type retourné est `string` (chaîne de caractères).

Ce genre de problèmes est très fréquent lors de la récupération des informations d'une base de données et il est nécessaire de bien mémoriser ce point.

La solution est simple et consiste à indiquer que la valeur est bien du type `integer` en préfixant la valeur avec (`int`).

```
$new_content = array('id' => (int) $current_breve[0]['id'], 'text'  
=> $current_breve[0]['text'], 'dt_creation' => $current_breve[0]  
['dt_creation'], );
```

Fig.21

Cette fois la valeur est bien un entier et nous pouvons donc créer l'objet:

```
$breve_to_update = new Breve($new_content);
```

Fig.22

Nous modifions le contenu de la brève avec la méthode dédiée `setText`:

```
$breve_to_update->setText('Nouveau contenu de la brève');
```

Fig.23

Puis nous procédons à la mise à jour de cette brève dans la base avec la méthode `updateBreve`:

```
$manager->updateBreve($breve_to_update);
```

Fig.24

Nous effectuons naturellement une vérification afin de contrôler que la mise à jour de la brève dans la base de données est réussie:

```
$check_breve = $manager->getBreve(2);  
var_dump($check_breve);
```

```
array  
0 =>  
array  
'id' => string '2' (length=1)  
'text' => string 'Nouveau contenu de la brève' (length=28)  
'dt_creation' => string '2013-10-14' (length=10)
```

Fig.25

Nous constatons que les opérations fondamentales **CRUD** sont bien fonctionnelles et pouvons donc implémenter ces deux classes dans une interface web conviviale afin que les créations, les lectures, les mises à jour et les suppressions puissent être opérées par un utilisateur.

Pour ce faire, nous opterons pour une structure de type **MVC** simple.