Classe SqlSimple

V1.3.1

Table des matières

1. Contenu

Ce document à pour but d'expliquer et de définir l'usage de la classe **`SqlSimple`** de l'outil **UniversalWeb**.

Pour obtenir la version de cette classe, tapez **`echo SqlSimple::VERSION`**.

2. Définition

La création de cette classe est issue du constat suivant : 60% des requêtes SQL d'un projet informatique sont constituées de requêtes standard d'insertion, modification ou suppression d'éléments souvent simples issus des tables de la base de données. La classe **`SqlSimple`** a donc pour tâche de :

- Simplifier l'appel des commandes SQL standards et d'opérations classiques pour le développeur évitant ainsi les éventuelles répétitions de code souvent génératrices de bugs.
- Permettre un gain de temps de développement d'applications basiques faisant appel à la base de données
- Assurer une qualité de service en proposant des requêtes pré-écrites fonctionnelles et éprouvées.

Cette classe s'applique facilement aux tables de configuration d'une application souvent formées de façon équivalentes, génériques…

Selon l'importance de l'opération à mener sur une table, il existe deux façon d'accéder aux données de la base : via les méthodes statiques de la classe et via les méthodes classiques instanciées. Extrêmement rapides à mettre en œuvre, les méthodes statiques vont permettre d'effectuer des opérations simples comme l'interrogation, voir la modification d'un champ d'une table, tandis que les méthodes instanciées sous forme d'objet vont permettre toutes sortes de manipulations plus complexes et efficaces.

3. Méthodes statiques

catalog

Cette méthode retourne un tableau contenant tous les couples **`$index`** / **`$champ`** de la table **`$table`**.

Description

```
mixed catalog(string $table, string $index, string $champ);
```

Liste des paramètres

`table` : nom de la table interrogée
`index` : premier champ à interroger servant d'index. Ce champ sert aussi de tri de présentation du tableau résultat.
`champ` : deuxième champ de correspondance

Valeurs de retour

`false` : erreur SQL.
tableau résultat si OK

Exemple
L'exemple suivant renvoie un tableau contenant tous les couples `titre` / `réalisateur` de la tables `films` triés par titres.

```
$liste = SqlSimple::catalog('films', 'titre', 'realisateur');
DEBUG_TAB_($liste);
```

Renvoie

```
Array
(
    [0] => Array

        (
            [titre] => 2001: l'odyssée de l'espace
            [realisateur] => Stanley Kubrick
        )

    [1] => Array
        (
            [titre] => Amityville, la maison du diable
            [realisateur] => Stuart Rosenberg
        )

    [2] => Array
        (
            [titre] => Autant en emporte le vent
            [realisateur] => Victor Fleming
etc.
```

distinct

Cette méthode retourne un tableau des valeurs uniques du champ **`$champ`** de la table **`$table`**.

Description

```
mixed distinct(string $table, string $champ [, string $tri]);
```

Liste des paramètres

`table` : nom de la table interrogée
`champ` : champ interrogé
`tri` : optionnel : champ de tri pour la présentation des résultats dans le tableau en retour

Valeurs de retour

`false` : erreur SQL.
tableau résultat si OK

Exemple
L'exemple suivant renvoie la liste uniques des années de production des films de la table `films` et triée par années.

```
$liste = SqlSimple::distinct('films', 'annee', 'annee');
DEBUG_TAB_($liste);
```

Renvoie

```
Array
(
    [0] => 1938
    [1] => 1939
    [2] => 1951
    [3] => 1952
    [4] => 1956
    [5] => 1962
    [6] => 1964
    [7] => 1966
    [8] => 1968
    [9] => 1977
    [10] => 1979
    [11] => 1980
)
```

existValeur

Cette méthode permet de rapidement savoir si **$valeur** est une valeur du champ **$champ** de la table **$table**.

Description

```
integer|boolean existValeur(string $table, string $field, string $valeur);
```

Liste des paramètres

table : nom de la table interrogée
field : nom du champ à interroger
valeur : valeur recherchée

Valeurs de retour

false : erreur SQL.
entier : nombre de fois où la valeur a été trouvée

Exemple

L'exemple suivant renvoie le nombre de clients dont le prénom est 'Francis'. Vous pouvez voir que l'on écrit aucune commande SQL !

```
$nb = SqlSimple::existValeur('clients', 'prenom', 'Francis');
```

existValeurAilleurs

Cette méthode permet de savoir si la table **$table** possède un champ **$field** à la valeur **$valeur** pour des tuples autres que celui dont la clé unique **$id** à la valeur **$valeurId**. Par exemple, la méthode est utilisée pour savoir si une valeur est déjà présente pour un identifiant autre que celui en cours. On peut ainsi vérifier en avance de phase qu'une valeur à ajouter sera bien unique.

Description

```
mixed existValeurAilleurs(string $table, string $field, string $valeur,
string $id, string $valeurId);
```

Liste des paramètres

table : nom de la table interrogée
field : nom du champ à interroger
valeur : valeur recherchée
id : champ unique de la table
valeurId : valeur de la clé unique d'exception

Valeurs de retour

false : erreur SQL.
entier : nombre de fois où la valeur a été trouvée

Exemple

L'exemple suivant renvoie combien de fois l'email **jean.serien@domaine.com** est trouvé dans la table **clients** pour les utilisateurs autres que celui dont l'identifiant est **322**.

```
$nb = SqlSimple::existValeurAilleurs('clients', 'email', 'jean.serien@domaine.com', 'id_client', '322');
```

L'exemple suivant teste si le libelle **stylos** existe déjà dans la table **fournitures** pour un identifiant de fournitures autre que le numéro **102**. Utile pour vérifier si l'éventuel ajout d'un libellé **stylos** ne risque pas par exemple de lever une erreur de clé dupliquée dans le cas d'un champ

libelle unique.

```
$nb = SqlSimple::existValeurAilleurs('fournitures', 'libelle', 'stylos', 'id_fourniture', '102');
```

fillSelect

Construit le code HTML interne d'un tag **<select>** d'une liste déroulante présentant tous les couples **$indice** / **$libelle** pour la table **$table**. Très utile pour proposer tout le contenu d'une table de référence.

Description

```
string fillSelect(mixed $defaut, string $table, string $indice, string $libelle);
```

Liste des paramètres

defaut : indice par défaut à afficher sur la liste déroulante (peut être un entier ou une chaîne de caractères).
table : nom de la table à partir de laquelle puiser les données.
indice : champ de la table contenant les indices de la liste correspondants aux libellés à afficher.
libelle : champ de la table contenant les libellés à afficher en correspondance de chaque indice.

Valeurs de retour

Le code HTML pour la liste déroulante.

Exemple

L'exemple suivant crée et affiche une liste déroulante contenant tous les genres de films contenus dans la table **genres**. Elle affichera par défaut le genre cinématographique dont l'identifiant est 2.

```
$html = '<select>';
$html.= SqlSimple::fillSelect(2, 'genres', 'id_genre', 'libelle_genre');
$html.= '</select>';
echo $html;
```

getGap

Renvoie le premier trou numérique dans le champ numérique **$champ** de la table **$table.**

Description

```
numeric getGap(string $table, string $champ [, numeric $sauf]);
```

Liste des paramètres

table : nom de la table à partir de laquelle puiser les données.
champ : champ interrogé.
sauf : optionnelle : Si cette valeur est positionnée, alors le trou sera recherché seulement à partir de cette valeur.

Valeurs de retour

La méthode renvoie la valeur directement inférieure à la valeur minimum du champ si celle-ci n'existe pas. Elle envoie la valeur directement supérieure à la valeur maximum si aucun trou n'est trouvé.
Le spectre de réponse de la méthode est [0 .. MAX + 1], cela signifie que la méthode n'envoie pas de valeur négative, la plus basse proposable étant 0. De préférence, à utiliser avec des entiers.

Exemple

L'exemple suivant renverra la valeur 1937, la plus petite année étant 1938.

```
$liste = array(1938, 1939, 1951, 1952, 1956, 1962, 1964, 1966, 1968, 1977, 1979, 1980);
$val = SqlSimple::getGap('films', 'annee');
echo $val;
1937
```

Cet autre exemple renverra 1950 :

```
$liste = array(1938, 1939, 1951, 1952, 1956, 1962, 1964, 1966, 1968, 1977, 1979, 1980);
$val = SqlSimple::getGap('films', 'annee', 1937);
echo $val;
1950
```

getMax

Renvoie la plus grande valeur du champ numérique **$champ** de la table **$table** éventuellement augmenté de la valeur **$offset**.

Cette méthode revêt un intérêt particulier si vous gérez par vous-même les clés primaires d'une table sans utiliser l'auto gestion AUTO-INCREMENT.

Description

```
numeric getMax(string $table, string $champ [, numeric $offset]);
```

Liste des paramètres

table : nom de la table à partir de laquelle puiser les données.
champ : champ interrogé.
offset : optionnel : valeur à ajouter au résultat de la requête.

Valeurs de retour

La valeur maximum du champ augmentée de **$offset**.

Exemple

L'exemple suivant renverra la valeur 1990.

```
$liste = array(1938, 1939, 1951, 1952, 1956, 1962, 1964, 1966, 1968, 1977, 1979, 1980);
$val = SqlSimple::getMax('films', 'annee', 10);
echo $val;
1990
```

getMin

Renvoie la plus petite valeur du champ numérique **$champ** de la table **$table** éventuellement augmenté de la valeur **$offset**.

Description

```
numeric getMin(string $table, string $champ [, numeric $offset]);
```

Liste des paramètres

table : nom de la table à partir de laquelle puiser les données.
champ : champ interrogé.
offset : optionnel : valeur à ajouter au résultat de la requête.

Valeurs de retour

La valeur minimum du champ augmentée de **$offset**.

Exemple

L'exemple suivant renverra la valeur 1948, la plus petite année étant 1938.

```
$liste = array(1938, 1939, 1951, 1952, 1956, 1962, 1964, 1966, 1968, 1977, 1979, 1980);
$val = SqlSimple::getMin('films', 'annee', 10);
echo $val;
1948
```

getValeurForKey

Permet de rechercher la valeur d'un champ pour une clé unique donnée.
Renvoie la valeur du champ **$field** de la table **$table** pour laquelle le champ unique **$key** a la valeur **$valeur**.

Description

```
mixed getValeurForKey(string $table, string $field, string $key, string $valeur [, boolean $debug]);
```

Liste des paramètres

table : nom de la table interrogée
field : nom du champ à interroger
key : champ unique de la table sur lequel se base la recherche
valeur : valeur de la clé unique
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

valeur du champ **field** pour lequel la clé unique **key** a la valeur **valeur**.
true : méthode en mode débogage.
false : aucune valeur trouvée **ou** **key** n'est pas une clé unique **ou** erreur SQL.

Exemple

L'exemple suivant renvoie le prénom du client dont l'identifiant est **322**.

```
$prenom = SqlSimple::getValeurForKey('clients', 'prenom', 'id_client', '322');
```

swapBool

Intervertit la valeur d'un champ booléen pour une clé unique donnée.
Intervertit la valeur du champ booléen **$field** de la table **$table** pour laquelle le champ unique **$key** a la valeur **$valeur**.
Après exécution de la méthode, si le champ contenait la valeur 1, il contiendra la valeur 0.
Après exécution de la méthode, si le champ contenait la valeur 0, il contiendra la valeur 1.

Description

```
mixed swapBool(string $table, string $champ, string $key, string $valeur [, boolean $debug]);
```

Liste des paramètres

table : nom de la table interrogée
champ : nom du champ à modifier
key : champ unique de la table sur lequel se base la recherche
valeur : valeur de la clé unique
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour
nombre de modifications effectuées (forcément 1).
false : erreur SQL.

Exemple
L'exemple modifie le flag **vote** pour l'utilisateur dont l'identifiant est **322**.

```
$prenom = SqlSimple::swapBool('clients', 'vote', 'id_client', '322');
```

updateChamp

Modification du champ **$champ** de la table **$table**. Lui est donnée la valeur **$valeur** pour l'identifiant de clé unique **$key** possédant la valeur **$id**.

Description

```
mixed updateChamp(string $table, string $champ, string $valeur, string $key, string $id [, boolean $debug]);
```

Liste des paramètres
table : nom de la table à modifier
champ : nom du champ à modifier
valeur : valeur à donner au champ
key : clé unique à laquelle faire référence
id : valeur de la clé unique pour distinguer la donne donnée
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour
nombre de tuples modifiés.
false : erreur SQL.

Exemple
Voici comment attribuer faussement un film à un réalisateur !

```
$res = SqlSimple::updateChamp('films', 'realisateur', 'George Lucas', 'titre', 'La planète des singes');
```

4. Méthodes publiques

Les méthodes statiques ci-dessus représentent le seul moyen d'utiliser directement la classe **SqlSimple**. Pour accéder aux autres méthodes de la classe il est obligatoire de créer une classe dérivée à partir de laquelle le développeur pourra appeler les méthodes parents.

```
class SqlFilms extends SqlSimple {
}
```

La première chose à écrire ensuite est à renseigner 3 propriétés publiques qui font le lien avec la table à étudier :

```
class SqlFilms extends SqlSimple {
```

```
    public $table = 'films';
    public $index = 'titre';
    public $champs = 'titre, annee, realisateur, visuel, genre';
}
```

avec

$table : nom de la table de la base de donnees
$index : index unique de la table
$champs : liste des champs de la table

Le développeur pourra ensuite écrire ses propres méthodes en s'appuyant sur les méthodes parents suivantes de la classe :

add

Ajout d'un enregistrement à la table

Description

```
mixed add(string $chaine [, boolean $debug]);
```

Liste des paramètres

chaine : chaine de caractère contenant la portion de code SQL d'ajout des données. La chaine de caractère doit être échappée.
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

nombre de tuples insérés.
false : erreur SQL.

Exemple

L'exemple suivant montre comment écrire une méthode héritée de la méthode **add** afin d'ajouter un tuple. Cette méthode va passer à la méthode parent le seul code SQL d'insertion à exécuter. Ici le choix a été fait de passer en entrée de notre méthode les données via un tableau. Celui-ci peut être le résultat d'un formulaire directement issu de **UniversalForm**.
Il est seulement nécessaire de construire ici l'affectation des champs et des données, le reste de la requête est construit par la méthode **add** parente (de la classe **SqlSimple** donc !)

```
class SqlFilms extends SqlSimple {
    public $table = 'films';
    public $index = 'titre';
    public $champs = 'titre, annee, realisateur, visuel, genre';

    public function add($donnees, $debug = false) {
        //ajouter le code Sql des champs nécéssaires pour l'ajout de données
        $requete = "'".$donnees['titre']."', ";
        $requete.= "'".$donnees['annee']."', ";
        $requete.= "'".$donnees['realisateur']."', ";
        $requete.= "'".$donnees['visuel']."', ";
        $requete.= "'".$donnees['genre']."' ";
        return parent::add($requete, $debug);
    }
}
```

addMany

Ajout de plusieurs tuples en une seule requête.

Description

```
mixed addMany(array $tabDonnees [, boolean $debug]);
```

Liste des paramètres

tabDonnees : tableau des données à insérer. Ce tableau doit contenir les tuples à insérer. **ATTENTION**, la structures des données doit correspondre strictement à la liste des champs attendus de la table (et dans l'ordre des champs). Les données doivent être échappées.
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

Nombre de tuples insérés si ok.
false : erreur SQL.

Exemple

```
class SqlFilms extends SqlSimple {
   public $table = 'films';
   public $index = 'titre';
   public $champs = 'titre, annee, realisateur, visuel, genre';
}

$tableFilms = new SqlFilms();
$lesFilms = array() ;
$unFilm = array('titre' => 'La guerre des étoiles',
                'annee' => '1977',
                'realisateur' => 'George Lucas',
                'titre' => 'Couleur',
                'genre' => 'science-fiction') ;
$lesFilms[] = $unFilm ;
$unFilm = array('titre' => 'Alien le 8ème passager',
                'annee' => '1978',
                'realisateur' => 'Ridley Scott',
                'titre' => 'Couleur',
                'genre' => 'science-fiction') ;
$lesFilms[] = $unFilm ;
$nbInsertions = $tableFilms->addMany($lesFilms);
if ($nbInsertions !== false) {
  echo $nbInsertions.' titres ajoutés';
}
else echo 'Erreur SQL';
```

delete

Suppression d'un enregistrement de la table

Description

```
mixed delete(string $id [, boolean $debug]);
```

Liste des paramètres

id : valeur de la clé primaire du tuple à supprimer.
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

nombre de tuples supprimés.
false : erreur SQL.

Exemple

```
class SqlFilms extends SqlSimple {
   public $table = 'films';
   public $index = 'titre';
   public $champs = 'titre, annee, realisateur, visuel, genre';
```

```
}

$tableFilms = new SqlFilms();
$nbDeleted = $tableFilms->delete('Vol au-dessus d\'un nid de coucous');
```

get

Retourne dans le tableau **$tuple** le tuple dont l'identifiant est **$id**.

Description

```
boolean get(string $id, array &$tuple [, boolean $debug]);
```

Liste des paramètres

id : identifiant primaire du tuple à retourner.
tuple : tuple retourné.
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

true : tuple trouvé.
false : tuple non trouvé.

Exemple

```
class SqlFilms extends SqlSimple {
   public $table = 'films';
   public $index = 'titre';
   public $champs = 'titre, annee, realisateur, visuel, genre';
}

$tableFilms = new SqlFilms();
$retour = $tableFilms->get('La guerre des étoiles', $leFilm);
if ($retour) {
  echo '<pre>';
  print_r($leFilm);
  echo '</pre>';
}
else echo 'Titre non trouvé';
```

getListeNombre

Retourne le nombre de tuples de la table

Description

```
mixed getListeNombre([boolean $debug]);
```

Liste des paramètres

debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

nombre de tuples dans la table.
false : erreur SQL.

Exemple

Ici il n'est pas nécessaire de surcharger la méthode depuis notre classe héritée car il n'y a aucune donnée à passer. On peut donc faire un appel direct à la méthode pour obtenir le nombre de tuples dans la table. Bien entendu, il est quand même nécessaire de faire cet appel par l'intermédiaire de

notre classe héritée **sqlFilms** qui possède les propriétés propres à la table désignée.

```
class SqlFilms extends SqlSimple {
   public $table = 'films';
   public $index = 'titre';
   public $champs = 'titre, annee, realisateur, visuel, genre';
}

$tableFilms = new SqlFilms();
$nbTuples = $tableFilms->getListeNombre();
```

getListe

Retourne un certain nombre de tuples de la table.
Retourne dans le tableau **$laListe**, **$nb_lignes** tuples à partir du tuple **$start**. Les données sont triées selon le champ **$tri** dans le sens **$sens**.

Description

```
mixed getListe(string $tri, string $sens, integer $start, integer
$nb_lignes, array &$laListe[, boolean $debug]);
```

Liste des paramètres

tri : champ sur lequel on souhaite que soit trié les données restituées
sens : sens de l'affichage souhaité (ASC / DESC)
start : tuple (ligne) de début de restitution
nb_lignes : nombre de lignes max à ramener
laListe : liste des tuples ramenés
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

nombre de tuples retournés.
false : erreur SQL.

Exemple

```
class SqlFilms extends SqlSimple {
   public $table = 'films';
   public $index = 'titre';
   public $champs = 'titre, annee, realisateur, visuel, genre';
}

$tableFilms = new SqlFilms();
$nbTuples = $tableFilms->getListeNombre();
$nbTuplesSelectionnes = $tableFilms->getListe('titre', 'ASC', 1, 25, $laListe);
foreach($laListe as $tuple) {
  echo $tuple['titre']. ' (. $tuple['annee'].')<br />';
}
```

getSome

Retourne tous les tuples de la table pour laquelle le champ **$champ** à la valeur **$valeur**.

Description

```
array getSome(string $champ, string $valeur, string $tri, array &$laListe
[, boolean $debug]);
```

Liste des paramètres

champ : champ sur lequel on effectue la recherche

valeur : valeur recherchée du champ
tri : champ sur lequel on souhaite que soit trié les données restituées (tri ascendant)
laListe : liste des tuples ramenés
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour
nombre de tuples retournés.
false : erreur SQL.

Exemple
L'exemple ci-dessous liste tous les films de George Lucas

```
class SqlFilms extends SqlSimple {
   public $table = 'films';
   public $index = 'titre';
   public $champs = 'titre, annee, realisateur, visuel, genre';
}

$tableFilms = new SqlFilms();
$nbTuplesSelectionnes = $tableFilms->getSome('realisateur', 'George Lucas', 'annee', $laListe);
foreach($laListe as $tuple) {
  echo $tuple['titre']. ' (' . $tuple['annee'].')<br />';
}
```

importMany

Insertion de plusieurs tuples en plusieurs requêtes. A la différence de la méthode **addMany**, **importMany** réalise plusieurs requêtes d'insertion SQL. C'est la méthode à adopter pour l'import en masse.
L'ordre des données disponibles dans le tableau d'entrée **$donnes** n'a pas d'importance. Seul le masque SQL doit correspondre aux champs (et à l'ordre des champs) attendus définis dans la propriété **$champs** de le classe héritée.

Description

```
mixed importMany(string $masqueSql, array $donnees [, integer $nb_tuple_par_requete][, boolean $debug]);
```

Liste des paramètres
masqueSql : masque de la requête SQL possédant des placeholders à la place des données à insérer. Chaque placeholders est constitué d'un numéro de rang entouré de crochets (ex : [4]), le numéro de rang correspondant à l'emplacement de la données réelle (indice) dans le tableau de données.

donnees : tableau des données à insérer. Attention contrairement aux autres méthodes d'insertion (**add**, **update**, **addMany**), les données ne doivent pas être échappées. C'est la méthode qui échappe automatiquement les données (voir exemple ci-dessous avec le titre « Vol au-dessus d'un nid de coucous »).

bb_tuple_par_requete : nombre optionnel de tuples max à insérer par requête (par défaut 10).

debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour
Nombre de tuples insérés si ok.

`false` : erreur SQL.

Exemple

Soit à insérer en masse le tableau de données suivantes :

```
$donnees = (tableau)
Array
(
    [0] => Array
        (
            [ligne] => 2
            [titre] => Vol au-dessus d'un nid de coucous
            [realisateur] => Milos Forman
            [visuel] => 1
            [annee] => 1975
            [genre] => drame
        )
    [1] => Array
        (
            [ligne] => 3
            [titre] => Terminator
            [realisateur] => James Cameron
            [visuel] => 1
            [annee] => 1984
            [genre] => science-fiction
        )
    [2] => Array
        (
            [ligne] => 4
            [titre] => Abyss
            [realisateur] => James Cameron
            [visuel] => 1
            [annee] => 1989
            [genre] => science-fiction
        )
)
```

Le masque correspondants doit être le suivant :

```
$leMasque = '[1], [4], [2], [3], [5]';
```

Au regard de la liste des champs (propriété **`$champs`**) définis pour la table :

```
class SqlFilms extends SqlSimple {
   public $table = 'films';
   public $index = 'titre';
   public $champs = 'titre, annee, realisateur, visuel, genre';
}
```

Le code d'insertion peut alors ainsi s'écrire :

```
class SqlFilms extends SqlSimple {
   public $table = 'films';
   public $index = 'titre';
   public $champs = 'titre, annee, realisateur, visuel, genre';
}

$tableFilms = new SqlFilms();
$leMasque = '[1], [4], [2], [3], [5]';
$nbInsertions = $tableFilms->importMany($leMasque, $donnees, 2);
```

Et qui produira le code SQL suivant :

```
INSERT IGNORE INTO films (titre, annee, realisateur, visuel, genre) VALUES
('Vol au-dessus d\'un nid de coucous', '1975', 'Milos Forman', '1', 'drame'),
('Terminator', '1984', 'James Cameron', '1', 'science-fiction')

INSERT IGNORE INTO films (titre, annee, realisateur, visuel, genre) VALUES
('Abyss', '1989', 'James Cameron', '1', 'science-fiction')
```

transaction

Exécute plusieurs requêtes fournies dans le tableau `$requetes` tout en opérant une transaction afin de garantir l'intégrité de la base de données. Intervertit la valeur d'un champ booléen pour une clé unique donnée.

Attention, cette méthode est efficace seulement sur les tables utilisant le moteur innoDB. En effet, les transactions ne sont pas prises en compte pour les tables ISAM.

Description

```
mixed transaction(array $requetes [, boolean $debug]);
```

Liste des paramètres

requetes : tableau contenant une suite de requêtes à exécuter dans la transaction
debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

0 : aucune modification n'est intervenue sur la base de données
true : la transaction s'est bien déroulée, toutes les requêtes ont été exécutées avec des résultats
false : erreur SQL.

Exemple

L'exemple suivant joue sur 2 tables : une table (`pays`) de regroupant des pays et une table regroupant des villes (`villes`). L'exemple suivant cherche à ajouter un pays et une ville dans chacune de ces tables. La table `villes` possède comme clé étrangère l'identifiant `idPays` du pays inséré dans la table `pays`.

```
public function addVille($donnees, $debug=false) {
    //requete d'insertion première table
    $requetes[1] = "INSERT IGNORE INTO pays ";
    $requetes[1].= "(idPays, nomPays) VALUES (";
    $requetes[1].= "NULL, "'".$donnees['nomPays']."'";
    $requetes[1].= ")";
    //requete d'insertion deuxième table
    $requetes[2] = "INSERT IGNORE INTO villes ";
    $requetes[2].= "(idVille, idPays, nomVille) VALUES (";
    $requetes[2].= "NULL, 'LAST_INSERT_ID', '".$donnees['nomVille']."'";
    $requetes[2].= ")";
    return parent::transaction($requetes, $debug);
}
```

Puis appel à la méthode `transaction` qui va ouvrir une transaction, exécuter les deux requêtes tout en veillant à la cohérence de la base de données : si l'une des 2 opérations d'insertion a échoué, la base de données va alors effectuer un `rollback` permettant de revenir à l'état d'avant la tentative d'insertion. La base de données est ainsi protégée.

On remarquera la valeur `LAST_INSERT_ID` dans la deuxième requête. Si besoin, il faut effectivement passer cette constante pour dire à la transaction que à partir de la deuxième table, on pourra utiliser l'id d'insertion créé dans la première table. `LAST_INSERT_ID` fait toujours référence à l'id d'insertion de la première table. Il faudra donc veiller à l'enchainement cohérent de vos requêtes.

On remarquera aussi que les requêtes sont indexées par ordre d'exécution choisi.

Voici un autre exemple d'utilisation : la suppression en cascade. Par exemple, en supprimant un pays on supprime aussi les villes de ce pays. La base de données reste cohérente même en cas d'erreur SQL sur l'une des tables.

```
class SqlPays extends SqlSimple {
```

```
    public function delPays($idPays, $debug=false) {
        $requetes[1] = "DELETE FROM villes WHERE idPays = 'idPays'";
        $requetes[1] = "DELETE FROM pays WHERE idPays = 'idPays'";
        return parent::transaction($requetes, $debug);
    }
}
```

update

Modification d'un enregistrement de la table

Description

```
mixed update(string $id, string $chaine [, boolean $debug]);
```

Liste des paramètres

id : valeur de la clé primaire du tuple à modifier.

chaine : chaine de caractère contenant la portion de code SQL de modification des données. La chaine de caractère doit être échappée.

debug : booléen de débogage. Si **true**, la requête n'est pas lancée mais le code SQL est affiché et la méthode renvoie **true**. Si **false** (valeur par défaut), la requête est exécutée.

Valeurs de retour

nombre de tuples modifiés.

false : erreur SQL.

Exemple

L'exemple suivant montre comment écrire une méthode héritée de la méthode **update** afin de modifier un tuple. Il est seulement nécessaire de construire ici l'affectation des champs et des données, le reste de la requête est construit par la méthode **update** parente. Ici le choix a été fait de passer en entrée de notre méthode les données via un tableau. Celui-ci peut être le résultat d'un formulaire directement issu de **UniversalForm**.

```
class SqlFilms extends SqlSimple {
    public $table = 'films';
    public $index = 'titre';
    public $champs = 'titre, annee, realisateur, visuel, genre';

    public function update($id, $donnees, $debug = false) {
        //ajouter le code Sql des champs nécéssaires pour l'ajout de données
        $requete = "titre = '".$donnees['titre']."', ";
        $requete.= "annee = '".$donnees['annee']."', ";
        $requete.= "realisateur = '".$donnees['realisateur']."', ";
        $requete.= "visuel = '".$donnees['visuel']."', ";
        $requete.= "genre = '".$donnees['genre']."' ";
        return parent::update($id, $requete, $debug);
    }
}
```