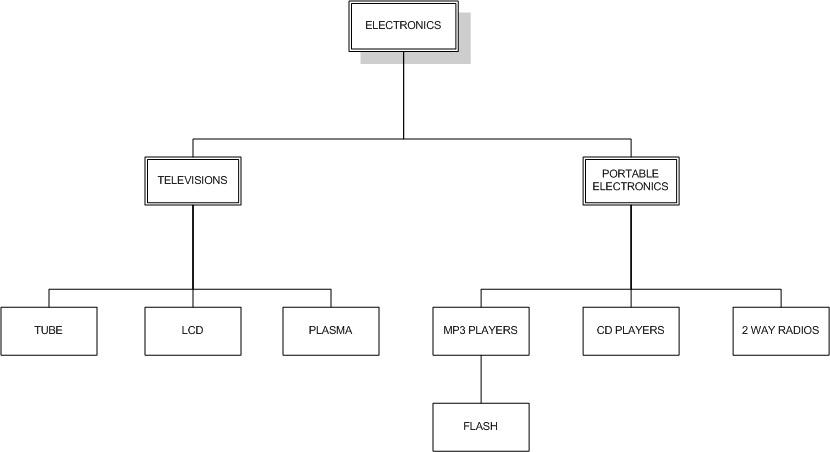
**UniversalTree**

Ce document à pour but de mettre en œuvre la classe UniversalTree.

# Qu’est-ce que la classe UniversalTree

Tout développeur à un moment donné est confronté avec la gestion de hiérarchies dans les bases de donnees et en particulier en SQL. Or les bases de données relationnelles ne permettent pas de résoudre facilement ce type de problème. Les tables d’une base de données relationnelle ne sont pas hiérarchiques (comme le XML) mais sont tout simplement une représentation à plat de données. Les données hiérarchiques possèdent quand à elles des relations parent-enfant qui ne sont pas représentées naturellement dans les tables de bases de donnees relationnelles.



Si l’on considère par exemple la hiérarchie ci-dessus, le SQL classique nécessaire à la récupération d’informations s’avère particulièrement délicat, ne serait-ce que pour représenter une telle hiérarchie à partir de données d’une table. Ici un exemple de requête nécessaire à obtenir ce type d’information.

SELECT t1.name AS lev1, t2.name as lev2, t3.name as lev3, t4.name as lev4

FROM category AS t1

LEFT JOIN category AS t2 ON t2.parent = t1.category\_id

LEFT JOIN category AS t3 ON t3.parent = t2.category\_id

LEFT JOIN category AS t4 ON t4.parent = t3.category\_id

WHERE t1.name = 'ELECTRONICS';

+-------------+----------------------+--------------+-------+

| lev1 | lev2 | lev3 | lev4 |

+-------------+----------------------+--------------+-------+

| ELECTRONICS | TELEVISIONS | TUBE | NULL |

| ELECTRONICS | TELEVISIONS | LCD | NULL |

| ELECTRONICS | TELEVISIONS | PLASMA | NULL |

| ELECTRONICS | PORTABLE ELECTRONICS | MP3 PLAYERS | FLASH |

| ELECTRONICS | PORTABLE ELECTRONICS | CD PLAYERS | NULL |

| ELECTRONICS | PORTABLE ELECTRONICS | 2 WAY RADIOS | NULL |

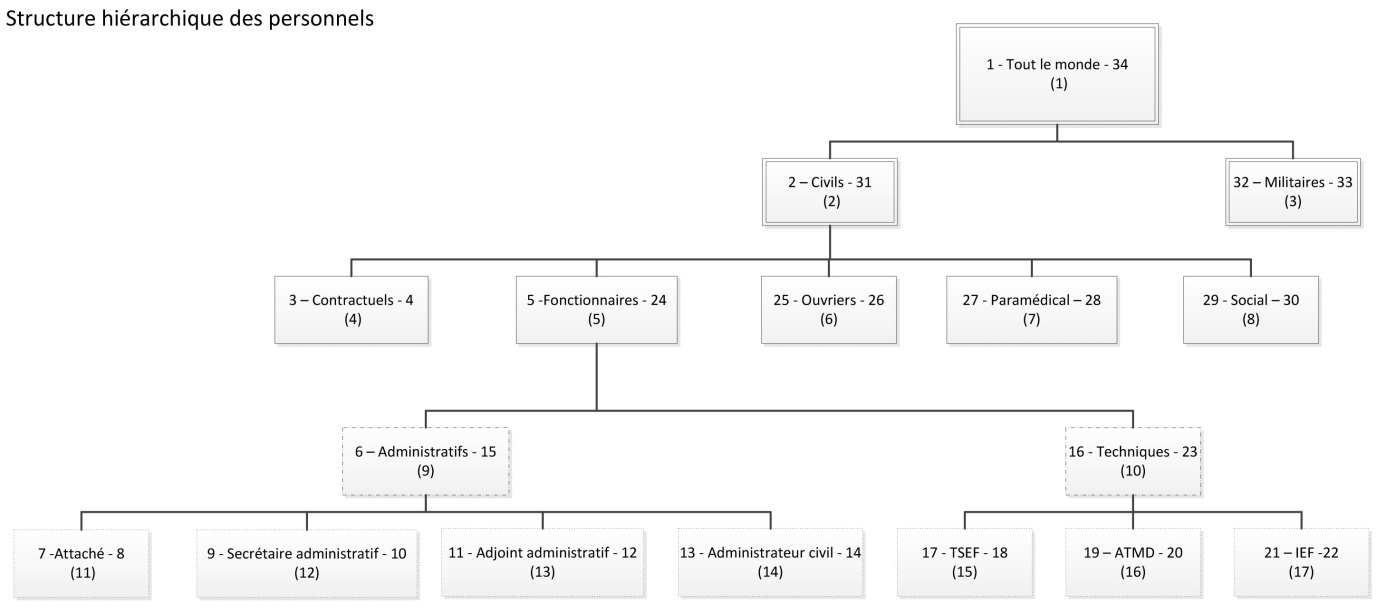
+-------------+----------------------+--------------+-------+

Fort heureusement il existe une autre méthode pour appréhender une hiérarchie, il s’agit de la méthode dites des « intervalles ». Je ne vais pas rentrer ici dans les détails de cette méthode, la page internet suivante le fait très bien <http://mikehillyer.com/articles/managing-hierarchical-data-in-mysql/>. Disons simplement qu’elle permet de voir un arbre en travaillant sur lui de gauche à droite, une couche à la suite de l’autre.

La classe UniversalTree permet donc de gérer des arbres hiérarchiques en utilisant la méthode des intervalles très économe en SQL.

# Construction d’une hiérarchie par l’exemple

Pour illustrer l’utilisation de cette classe et de ses méthodes, nous allons prendre un exemple : celui d’une structure hiérarchique de personnels que nous allons construire.



Comme le montre le schéma de cette organisation hiérarchique, la méthode des intervalles demande à ce que chaque « feuille » (élément) possède un identifiant gauche et un identifiant droite. Cette information n’est pas nécessaire pour mettre en œuvre cette hiérarchie dans une table SQL, mais je la présente là uniquement pour la compréhension de la méthode des intervalles.

Dans cet exemple, chaque feuille va représenter un élément de l’arbre hiérarchique. Pour l’exemple, l’élément d’indice 3 aura comme libellé « Militaires ».

## Création de l’arbre dans une table SQL

### Création de l’objet hiérarchique

La première chose à faire est de créer notre classe qui doit hériter de la classe UniversalTree.

class **TreePersonnels** extends **UniversalTree** {

}

### Création de la table SQL

Il faut maintenant créer la table SQL qui va recevoir les données de la représentation hiérarchique.

$tree = new **TreePersonnels**();

$tree->**setTable**('categories\_personnels');

$tree->**setKey**('id\_categorie\_personnel');

$tree->**createTable**();

Si l’on y regarde de plus près, la table categories\_personnels ainsi créée via la méthode createTable() a implémenté les champs suivants :

* **id\_categorie\_personnel** : id unique de chaque feuille de la hiérarchie. C’est la clé primaire de notre table qui permettra d’identifier de manière unique la feuille de notre hiérarchie. Le choix du nom (totalement libre) a été passé à la méthode setKey().
* **gauche** : entier qui contient la valeur gauche de la feuille. Ce champ est obligatoire. Il n’est pas directement géré par le programmeur mais c’est la classe UniversalTree qui le gère et l’utilise. Il ne faut pas changer le nom de ce champ.
* **droite** : entier qui contient la valeur droite de la feuille. De même, ce champ est obligatoire. Il n’est de même pas directement géré par le programmeur mais c’est la classe UniversalTree qui le gère et l’utilise.
* **niveau** : entier qui contient le niveau hiérarchique de la feuille dans l’arbre. Là encore, ce champ est obligatoire. Il n’est pas non plus directement géré par le programmeur mais c’est la classe UniversalTree qui le gère et l’utilise. Le niveau le plus haut (la racine de l’arbre) correspond au niveau 0 (zéro).

Reste au développeur à ajouter à la table categories\_personnels les champs de données propres associés à chaque feuille. Dans notre exemple, nous allons simplement ajouter le champ texte « libelle » qui va contenir le libellé de la catégorie de personnel.

Notons que l’appel aux méthodes setTable() et setKey() qui, respectivement, permettent d’associer à notre représentation hiérarchique le nom de la table SQL et le champ identifiant de la feuille (catégorie de personnel), pourrait très bien être intégré au niveau du constructeur de la classe TreePersonnels.

class **TreePersonnels** extends **UniversalTree** {

public function **\_\_construct**() {

$this->**setTable**('categories\_personnels');

$this->**setKey**('id\_categorie\_personnel');

}

}

### Création de la racine de l’arbre

La racine de l’arbre est représenté par la feuille libellé « Tout le monde ». On va lui donner comme id le nombre 1. La création d’une racine se fait via l’appel à la méthode addLeave(). On considère son niveau hiérarchique comme étant 0 (zéro).

//ajout racine

$item = array('id\_categorie\_personnel' => 1, 'libelle' => 'Tout le monde');

$tree->**addLeave**(0, $item);

La méthode reçoit 2 paramètre :

1. Le premier (un entier) correspond à l’identifiant de la feuille à laquelle la nouvelle feuille vient se raccorder. Dans le cas d’une feuille racine, l’arbre ne possédant encore aucune feuille, on peut entrer 0 (zéro).
2. Le second (un tableau array) contient les données de l’application à fournir pour le tuple.

### Création des feuilles du niveau 1

Nous allons maintenant créer les feuilles « Civils » et « Militaires » qui correspondent au niveau 1 de notre hiérarchie. C’est ici le moment de parler de la notion de nœud d’arborescence. Un nœud est le branchement d’une feuille à une feuille parente. C’est le cas de la feuille « Civils ». Pour créer cette feuille et la relier à la feuille « Tout le monde » il faut faire appel à la méthode addNode(). Le code est celui-ci :

//ajout niveau 1

$item = array('id\_categorie\_personnel' => 2, 'libelle' => 'Civils');

$tree->**addNode**(1, $item);

La syntaxe est identique à la méthode addLeave() si ce n’est qu’elle traite un nœud (ici donc une feuille à rattacher à une feuille parente).

1. Le premier paramètre (un entier) correspond à l’identifiant de la feuille parente à laquelle la nouvelle feuille vient se raccorder. Ici, nous lui spécifions la feuille identifiée par la valeur id\_categorie\_personnel = 1, qui est l’id de la catégorie de personnel de la feuille « Tout le monde ».
2. Le second, tout comme la méthode addLeave() est un tableau (array) contenant les données de l’application à fournir pour le tuple.

Pour l’ajout de la feuille « Militaires », nous allons faire appel à la méthode addLeave(). En effet, « Militaires » étant sur le même niveau que la feuille « Civils » que nous venons de créer, il s’agit bien ici de créer une nouvelle feuille (qui va se raccorder à la feuille « Civils ») et non un nouveau nœud. Le code complet nécessaire à la création du niveau 1 de notre hiérarchie est donc le suivant :

//ajout niveau 1

$item = array('id\_categorie\_personnel' => 2, 'libelle' => 'Civils');

$tree->**addNode**(1, $item);

$item = array('id\_categorie\_personnel' => 3, 'libelle' => 'Militaires');

$tree->**addLeave**(2, $item);

### Création des feuilles de niveau 2, 3 et 4

Pas de problème particulier, la création de ces niveaux, tout comme celle du niveau 1, est une succession d’appels aux méthodes addNode() et addLeave(). Il s’agit juste de garder à l’esprit quelle est la feuille qui a la particularité d’être un nœud.

//ajout niveau 2

$item = array('id\_categorie\_personnel' => 4, 'libelle' => 'Contractuel');

$tree->**addNode**(2, $item);

$item = array('id\_categorie\_personnel' => 5, 'libelle' => 'Fonctionnaires');

$tree->**addLeave**(4, $item);

$item = array('id\_categorie\_personnel' => 6, 'libelle' => 'Ouvriers');

$tree->**addLeave**(5, $item);

$item = array('id\_categorie\_personnel' => 7, 'libelle' => 'Paramédical');

$tree->**addLeave**(6, $item);

$item = array('id\_categorie\_personnel' => 8, 'libelle' => 'Social');

$tree->**addLeave**(7, $item);

//ajout niveau 3

$item = array('id\_categorie\_personnel' => 9, 'libelle' => 'Administratifs');

$tree->**addNode**(5, $item);

$item = array('id\_categorie\_personnel' => 10, 'libelle' => 'Techniques');

$tree->**addLeave**(9, $item);

//ajout niveau 4

$item = array('id\_categorie\_personnel' => 11, 'libelle' => 'Attaché');

$tree->**addNode**(9, $item);

$item = array('id\_categorie\_personnel' => 12, 'libelle' => 'Secrétaire Administratif');

$tree->**addLeave**(11, $item);

$item = array('id\_categorie\_personnel' => 13, 'libelle' => 'Adjoint Administratif');

$tree->**addLeave**(12, $item);

$item = array('id\_categorie\_personnel' => 14, 'libelle' => 'Administrateur Civil');

$tree->**addLeave**(13, $item);

$item = array('id\_categorie\_personnel' => 15, 'libelle' => 'TSEF');

$tree->**addNode**(10, $item);

$item = array('id\_categorie\_personnel' => 16, 'libelle' => 'ATMD');

$tree->**addLeave**(15, $item);

$item = array('id\_categorie\_personnel' => 17, 'libelle' => 'IEF');

$tree->**addLeave**(16, $item);

## Afficher la structure de l’arbre

Maintenant que nous avons créé notre hiérarchie de catégories de personnels dans la table categories\_personnels, voyons comment afficher cette arborescence.

//affichage de l'arbre complet créé

$res = $tree->**getTree**(0, false, false);

$tree->**display**('ARBRE COMPLET HIERARCHIQUE', $res);

La méthode getTree() permet de récupérer les informations de l’arbre. Elle possède 3 paramètres :

1. $id\_node : (entier) nœud à partir duquel explorer l’arbre hiérarchique. C’est le numéro de la feuille passé à la clé de la table id\_categorie\_personnel.
2. $nodeToo : (booléen). Si true, la méthode renvoie également la feuille parente. Si false, la méthode ne renvoie pas la feuille parente.
3. $byLevel : (booléen). Si true, la restitution de l’arborescence est faite par ordre de niveau, c’est-à-dire que chaque niveau est présenté lorsque le niveau supérieur est épuisé. Si false, la restitution de l’arborescence est faite par ordre hiérarchique, c’est-à-dire que la totalité d’une hiérarchie est explorée avant de passer à une autre. Ce paramètre est optionnel. Si non fourni, l’exploration de l’arbre est réalisé par ordre hiérarchique.

L’appel à la méthode display() affiche le résultat renvoyé sous forme de tableau par la méthode getTree(). C’est une méthode basique dites de débogage car elle ne permet que d’afficher les feuilles et leur identifiant. Il faudra la surcharger dans votre classe héritée (ici TreePersonnels) pour un meilleur rendu. Le résultat de l’affichage de notre arborescence par ordre hiérarchique grâce à la méthode display() est le suivant :

ARBRE COMPLET HIERARCHIQUE

1 (1)

… 2 (2)

…… 4 (4)

…… 5 (5)

……… 9 (9)

………… 11 (11)

………… 12 (12)

………… 13 (13)

………… 14 (14)

……… 10 (10)

………… 15 (15)

………… 16 (16)

………… 17 (17)

…… 6 (6)

…… 7 (7)

…… 8 (8)

… 3 (3)

On constate que toute la hiérarchie est explorée dans le sens vertical (ordre hiérarchique). Pour afficher notre arbre dans le sens horizontal (ordre de niveau), il aurait fallu écrire :

//affichage de l'arbre complet créé

$res = $tree->**getTree**(0, false, **true**);

$tree->**display**('ARBRE COMPLET HIERARCHIQUE', $res);

Ce qui aurait donné l’affichage suivant :

ARBRE COMPLET HIERARCHIQUE

1 (1)

… 2 (2)

… 3 (3)

…… 4 (4)

…… 5 (5)

…… 6 (6)

…… 7 (7)

…… 8 (8)

……… 10 (10)

……… 9 (9)

………… 11 (11)

………… 12 (12)

………… 13 (13)

………… 14 (14)

………… 15 (15)

………… 16 (16)

………… 17 (17)

Voici un exemple de surcharge de la méthode display() qui permet d’afficher le contenu réel de chaque feuille de notre arbre.

//----------------------------------------------

// Affichage du contenu de l'arbre

// Entree :

// $libelle : libellé libre à afficher

// $tableau : tableau d'éléments (feuilles et noeuds) de l'arbre

// Retour : écriture sur la sortie standard

//----------------------------------------------

public function **display**($libelle, $tableau) {

echo '<p><u>'.$libelle.'</u><br />';

**foreach**($tableau as $element) {

**for**($i = 0; $i < $element['niveau']; $i++) **echo** '&hellip;';

**echo** ' '.$element['libelle'].' ('.$element['id\_categorie\_personnel'].')<br />';

}

**echo** '</p>';

}

Pour le résultat suivant :

ARBRE COMPLET HIERARCHIQUE

Tout le monde (1)

… Civils (2)

…… Contractuel (4)

…… Fonctionnaires (5)

……… Administratifs (9)

………… Attaché (11)

………… Secrétaire Administratif (12)

………… Adjoint Administratif (13)

………… Administrateur Civil (14)

……… Techniques (10)

………… TSEF (15)

………… ATMD (16)

………… IEF (17)

…… Ouvriers (6)

…… Paramédical (7)

…… Social (8)

… Militaires (3)

Bien entendu on peut n’afficher qu’une partie de l’arbre. Il suffit pour cela de demander une restitution de l’arbre à partir d’une feuille en particulier. Par exemple l’arbre (on, parle alors de sous arbre) « Fonctionnaires » sera chargé ainsi :

//affichage de l'arbre des fionctionnaires

$res = $tree->**getTree**(**5**, false, false);

$tree->**display**('ARBRE FONCTIONNAIRES', $res);

ARBRE FONCTIONNAIRES

……… Administratifs (9)

………… Attaché (11)

………… Secrétaire Administratif (12)

………… Adjoint Administratif (13)

………… Administrateur Civil (14)

……… Techniques (10)

………… TSEF (15)

………… ATMD (16)

………… IEF (17)

En complément à la méthode getTree(), il existe plusieurs autres méthodes pour affiner les données à récupérer. Je vous renvoie aux annexes qui décrivent les méthodes concernées.

## Supprimer une feuille

Pour supprimer une feuille de l’arbre, on fait appel à la méthode delItem($id, $delChildren=true).

* $id : (entier) identifiant de l’élément à supprimer (pour notre exemple, l’identifiant id\_categorie\_personnel).
* $delChildren : (booléen). Si true, non seulement la feuille sera supprimée mais également toutes ses feuilles enfant. Si false, seule la feuille choisie est supprimée. Ce paramètre peut être omis. Dans ce cas, toutes les feuilles enfants sont aussi supprimées.

La suppression du sous arbre « Administratifs »

//Suppression du sous arbre "administratifs"

$res = $tree->**delItem**(9, true);

$res = $tree->**getTree**(0, false, false);

$tree->**display**('ARBRE COMPLET HIERARCHIQUE', $res);

Donnera le résultat suivant :

ARBRE COMPLET HIERARCHIQUE

Tout le monde (1)

… Civils (2)

…… Contractuel (4)

…… Fonctionnaires (5)

……… Techniques (10)

………… TSEF (15)

………… ATMD (16)

………… IEF (17)

…… Ouvriers (6)

…… Paramédical (7)

…… Social (8)

… Militaires (3)

La suppression de la feuille « Administratifs »

//Suppression du sous arbre "administratifs"

$res = $tree->**delItem**(9, false);

$res = $tree->**getTree**(0, false, false);

$tree->**display**('ARBRE COMPLET HIERARCHIQUE', $res);

Donnera le résultat suivant :

ARBRE COMPLET HIERARCHIQUE

Tout le monde (1)

… Civils (2)

…… Contractuel (4)

…… Fonctionnaires (5)

……… Attaché (11)

……… Secrétaire Administratif (12)

……… Adjoint Administratif (13)

……… Administrateur Civil (14)

……… Techniques (10)

………… TSEF (15)

………… ATMD (16)

………… IEF (17)

…… Ouvriers (6)

…… Paramédical (7)

…… Social (8)

… Militaires (3)

**Conséquence directe**, les feuilles enfants (s’il y en a) de la feuille supprimée sont directement rattachée à la feuille parente de la feuille supprimée. Attention donc à l’intégrité de votre hiérarchie !

On peut réinsérer la feuille « Administratifs »

//Re-ajout du neoud "Administratifs"

$item = array('id\_categorie\_personnel' => 9, 'libelle' => 'Administratifs');

$tree->**addNode**(5, $item);

$res = $tree->**getTree**(0, false, false);

$tree->**display**('ARBRE COMPLET HIERARCHIQUE', $res);

Mais attention, cette nouvelle insertion n’a pas réintégré les ancienne feuilles enfants. Elles sont maintenant au même niveau hiérarchique que la feuille insérée. On voit que cette méthode permet par exemple de remonter d’un niveau un sous arbre.

ARBRE COMPLET HIERARCHIQUE

Tout le monde (1)

… Civils (2)

…… Contractuel (4)

…… Fonctionnaires (5)

……… Administratifs (9)

……… Attaché (11)

……… Secrétaire Administratif (12)

……… Adjoint Administratif (13)

……… Administrateur Civil (14)

……… Techniques (10)

………… TSEF (15)

………… ATMD (16)

………… IEF (17)

…… Ouvriers (6)

…… Paramédical (7)

…… Social (8)

… Militaires (3)

# Annexes

## Méthodes publiques

|  |
| --- |
| setTable |
| Affecte à l’objet le nom de la table SQL qui sera utilisée pour stocker notre hiérarchie.  **Description**  **setTable**(**string** $table)  **Liste des paramètres**  **table**  **chaine de caractère contenant le nom de la table qui doit contenir la hiérarchie dans la base de données.**  **Exemple**  Soit la construction du filtre externe identifié ‘search’ :  public function **\_\_construct**() {  $this->**setTable**('categories\_personnels');  $this->setKey('id\_categorie\_personnel');  } |

|  |
| --- |
| setKey |
| Affecte à l’objet le nom du champ qui va servir à identifier de manière unique chaque feuille de l’arbre dans la table SQL.  **Description**  **setKey**(**string** $key)  **Liste des paramètres**  **key**  **chaine de caractère contenant le nom du champ qui va identifier les feuilles.**  **Exemple**  Soit la construction du filtre externe identifié ‘search’ :  public function **\_\_construct**() {  $this->setTable('categories\_personnels');  $this->**setKey**('id\_categorie\_personnel');  } |

|  |
| --- |
| createTable |
| Créé dans la base de données la table qui va recevoir l’arborescence hiérarchique.  **Description**  bool **createTable**()  **Liste des paramètres**  **Aucun**  **Retour**  true**: la création de la table s’est bien passé.**  false**: erreur lors de la création de la table**  Cette appel n’est à faire qu’une seule fois. Si la table existe déjà, la méthode ignore la demande.  L’appel aux méthodes setTable() et setKey() doit impérativement avoir été réalisé avant l’appel de la méthode createTable(). Dans le cas contraire, l’application sera stoppée et un message indiquera l’oubli. |

|  |
| --- |
| drop |
| Vide la table qui reçoit l’arborescence hiérarchique.  **Description**  bool **drop**()  **Liste des paramètres**  **Aucun**  **Retour**  true**: la table a été vidée avec succès.**  false**: erreur lors du vidage de la table** |

|  |
| --- |
| getRoot |
| Renvoie le nœud racine de l’arbre.  **Description**  array **getRoot**()  **Liste des paramètres**  **Aucun**  **Retour**  **Un tableau indicé par ‘0’ qui contient les données de la feuille racine. Si il existe plusieurs racines, alors la méthode renvoie un tableau vide. Si il y a une erreur SQL, la méthode renvoie** false**.**  **Exemple de retour :**  Array  (  [0] => Array (  [id\_categorie\_personnel] => 1  [gauche] => 1  [droite] => 34  [niveau] => 0  [libelle] => Tout le monde  )  ) |

|  |
| --- |
| getLeaves |
| Récupère toutes les feuilles de l’arbre ou d’un sous arbre.  **Description**  array **getLeaves**([int $id\_node] )  **Liste des paramètres**  **id\_node**  **Paramètre optionnel. Spécifie le nœud à partir duquel retrouver les feuilles de l’arborescence.**  **Retour**  **Un tableau indicé qui contient les feuilles.**  **Attention : Les nœuds ne sont pas récupérés. En fin de compte sont récupéré toutes les feuilles qui n’ont pas d’enfants.**  **Exemple : Récupérer toutes les feuilles « Fonctionnaires »**  **$root = $tree->getLeaves(5);**  Renvoie le tableau suivant :  Array  (  [0] => Array (  [id\_categorie\_personnel] => 11  [gauche] => 7  [droite] => 8  [niveau] => 4  [libelle] => Attaché  )  [1] => Array (  [id\_categorie\_personnel] => 12  [gauche] => 9  [droite] => 10  [niveau] => 4  [libelle] => Secrétaire Administratif  )  [2] => Array (  [id\_categorie\_personnel] => 13  [gauche] => 11  [droite] => 12  [niveau] => 4  [libelle] => Adjoint Administratif  )  [3] => Array (  [id\_categorie\_personnel] => 14  [gauche] => 13  [droite] => 14  [niveau] => 4  [libelle] => Administrateur Civil  )  [4] => Array (  [id\_categorie\_personnel] => 15  [gauche] => 17  [droite] => 18  [niveau] => 4  [libelle] => TSEF  )  [5] => Array (  [id\_categorie\_personnel] => 16  [gauche] => 19  [droite] => 20  [niveau] => 4  [libelle] => ATMD  )  [6] => Array (  [id\_categorie\_personnel] => 17  [gauche] => 21  [droite] => 22  [niveau] => 4  [libelle] => IEF  )  ) |

|  |
| --- |
| getNodes |
| Récupère toutes les nœuds de l’arbre ou d’un sous arbre.  **Description**  array **getNodes**([int $id\_node] )  **Liste des paramètres**  **id\_node**  **Paramètre optionnel. Spécifie le nœud à partir duquel retrouver les nœuds de l’arborescence.**  **Retour**  **Un tableau indicé qui contient les nœuds.**  **Attention : Les feuilles ne sont pas récupérés. En fin de compte sont récupéré toutes les feuilles qui ont des enfants.**  **Exemple : Récupérer tous les nœuds du sous arbre « Fonctionnaires »**  **$root = $tree->getNodes(5);**  Renvoie le tableau suivant :  Array  (  [0] => Array (  [id\_categorie\_personnel] => 9  [gauche] => 6  [droite] => 15  [niveau] => 3  [libelle] => Administratifs  )  [1] => Array (  [id\_categorie\_personnel] => 10  [gauche] => 16  [droite] => 23  [niveau] => 3  [libelle] => Techniques  )  ) |

|  |
| --- |
| getTree |
| Récupère toute l’arborescence de l’arbre ou d’un sous arbre.  **Description**  array **getTree**(int $id\_node, bool $nodeToo, [bool $byLevel])  **Liste des paramètres**  **id\_node**  **Spécifie le nœud à partir duquel retrouver l’arborescence.**  **nodeToo**  **Dit si la méthode doit aussi ramener le nœud parent**  true**: ramène le nœud parent.**  false**: ne ramène pas le nœud parent.**  **byLevel**  **Paramètre optionnel.**  true : la restitution de l’arborescence est faite par ordre de niveau, c’est-à-dire que chaque niveau est présenté lorsque le niveau supérieur est épuisé.  false : la restitution de l’arborescence est faite par ordre hiérarchique, c’est-à-dire que la totalité d’une hiérarchie est explorée avant de passer à une autre. (valeur par défaut)  **Retour**  **Un tableau indicé qui contient les feuilles et nœuds de l’arborescence.**  **Exemple : Récupérer le sous arbre « Fonctionnaires » y compris le nœud « Fonctionnaires »**  **$root = $tree->getTree(5, true, false);**  Renvoie le tableau suivant :  Array  (  [0] => Array (  [id\_categorie\_personnel] => 5  [gauche] => 5  [droite] => 24  [niveau] => 2  [libelle] => Fonctionnaires  )  [1] => Array (  [id\_categorie\_personnel] => 9  [gauche] => 6  [droite] => 15  [niveau] => 3  [libelle] => Administratifs  )  [2] => Array (  [id\_categorie\_personnel] => 11  [gauche] => 7  [droite] => 8  [niveau] => 4  [libelle] => Attaché  )  [3] => Array (  [id\_categorie\_personnel] => 12  [gauche] => 9  [droite] => 10  [niveau] => 4  [libelle] => Secrétaire Administratif  )  [4] => Array (  [id\_categorie\_personnel] => 13  [gauche] => 11  [droite] => 12  [niveau] => 4  [libelle] => Adjoint Administratif  )  [5] => Array (  [id\_categorie\_personnel] => 14  [gauche] => 13  [droite] => 14  [niveau] => 4  [libelle] => Administrateur Civil  )  [6] => Array (  [id\_categorie\_personnel] => 10  [gauche] => 16  [droite] => 23  [niveau] => 3  [libelle] => Techniques  )  [7] => Array (  [id\_categorie\_personnel] => 15  [gauche] => 17  [droite] => 18  [niveau] => 4  [libelle] => TSEF  )  [8] => Array (  [id\_categorie\_personnel] => 16  [gauche] => 19  [droite] => 20  [niveau] => 4  [libelle] => ATMD  )  [9] => Array (  [id\_categorie\_personnel] => 17  [gauche] => 21  [droite] => 22  [niveau] => 4  [libelle] => IEF  )  ) |

|  |
| --- |
| getTreeKeys |
| Récupère toutes les clés d’une feuille (ou d’un nœud) et de ses enfants et renvoie les informations d’arborescence par niveau.  **Description**  array **getTreeKeys**(int $id)  **Liste des paramètres**  **id**  **Spécifie la feuille ou le nœud à partir duquel retrouver les clés de l’arborescence.**  **Retour**  **Un tableau indicé qui contient uniquement les clés de l’arborescence.**  **Renvoie aussi la clé racine de l’arbre ou du sous arbre.**  **Exemple : Récupérer les clés du sous arbre « Fonctionnaires »**  **$root = $tree->getTreeKeys(5);**  Renvoie le tableau suivant :  Array  (  [0] => Array (  [id\_categorie\_personnel] => 5  )  [1] => Array (  [id\_categorie\_personnel] => 9  )  [2] => Array (  [id\_categorie\_personnel] => 10  )  [3] => Array (  [id\_categorie\_personnel] => 11  )  [4] => Array (  [id\_categorie\_personnel] => 12  )  [5] => Array (  [id\_categorie\_personnel] => 13  )  [6] => Array (  [id\_categorie\_personnel] => 14  )  [7] => Array (  [id\_categorie\_personnel] => 15  )  [8] => Array (  [id\_categorie\_personnel] => 16  )  [9] => Array (  [id\_categorie\_personnel] => 17  )  ) |

|  |
| --- |
| getTreeKeysList |
| Idem à la méthode getTreeKeys() mais renvoie les information sous forme de liste.  **Description**  array **getTreeKeysList**(int $id)  **Liste des paramètres**  **id**  **Spécifie la feuille ou le nœud à partir duquel retrouver les clés de l’arborescence.**  **Retour**  **Une liste des clés de l’arborescence séparées par une virgule.**  **Exemple : Récupérer les clés du sous arbre « Fonctionnaires »**  **$root = $tree->getTreeKeysList(5);**  Renvoie la liste suivante :  5,9,10,11,12,13,14,15,16,17 |

|  |
| --- |
| getParents |
| Récupère le chemin arborescent hiérarchique d’une nœud ou d’une feuille.  **Description**  array **getParents**(int $id\_node, [bool $nodeToo])  **Liste des paramètres**  **id\_node**  **Spécifie le nœud à partir duquel retrouver le chemin arborescent.**  **nodeToo**  **Paramètre optionnel. Dit si la méthode doit aussi ramener le nœud demandé.**  true**: ramène le nœud ou la feuille demandé.**  false**: ne ramène pas le nœud ou la feuille demandé.**  **Retour**  **Un tableau indicé qui contient les feuilles et nœuds représentant le chemin hiérarchique arborescent de l’élément.**  **Exemple : Récupérer le chemin hiérarchique arborescent de « Paramédical »**  **$chemin = $tree->getParents(7, true);**  Renvoie le tableau suivant :  Array  (  [0] => Array (  [id\_categorie\_personnel] => 1  [gauche] => 1  [droite] => 34  [niveau] => 0  [libelle] => Tout le monde  )  [1] => Array (  [id\_categorie\_personnel] => 2  [gauche] => 2  [droite] => 31  [niveau] => 1  [libelle] => Civils  )  [2] => Array (  [id\_categorie\_personnel] => 7  [gauche] => 27  [droite] => 28  [niveau] => 2  [libelle] => Paramédical  )  ) |

|  |
| --- |
| addLeave |
| Ajoute une feuille à l’arborescence.  **Description**  bool **addLeave**(int $id\_previous\_leave, array $infos)  **Liste des paramètres**  **id\_previous\_leave**  **id de la feuille précédente (gauche) dans l’arbre à laquelle rattacher la nouvelle feuille..**  **infos**  **Tableau associatif contenant pour chaque champ de la table une valeur à entrer.**  **Retour**  true**: l’insertion s’est bien passée.**  false**: erreur SQL**  **Exemple : Ajouter la feuille « Paramédical » à droite de la feuille « Ouvriers »**  **$item = array('id\_categorie\_personnel' => 7, 'libelle' => 'Paramédical');**  **$tree->addLeave(6, $item);** |

|  |
| --- |
| addNode |
| Ajoute un nœud à l’arborescence. Lors de la création du nœud (ou feuille) racine, c’est cette méthode qui doit être utilisée et non addLeave().  **Description**  bool **addNode**(int $id\_parent\_leave, array $infos)  **Liste des paramètres**  **id\_previous\_leave**  **id de la feuille dans l’arbre qui va devenir la feuille parente de la nouvelle feuille à rattacher.**  **infos**  **Tableau associatif contenant pour chaque champ de la table une valeur à entrer.**  **Retour**  true**: l’insertion s’est bien passée.**  false**: erreur SQL**  **Exemple : Ajouter le nœud « Administratifs » à la feuille « Fonctionnaires »**  **$item = array('id\_categorie\_personnel' => 9, 'libelle' => 'Administratifs');**  **$tree->addNode(5, $item);** |

|  |
| --- |
| delItem |
| Supprime un nœud ou une feuille de l’arborescence.  **Description**  bool **delItem**(int $id, bool $delChildren)  **Liste des paramètres**  **id**  **id du nœud ou de la feuille à retirer de l’arborescence.**  **delChildren**  Si true, non seulement la feuille sera supprimée mais également toutes ses feuilles enfant. Si false, seule la feuille choisie est supprimée. Ce paramètre peut être omis. Dans ce cas, toutes les feuilles enfants sont aussi supprimées.  **Retour**  true**: l’insertion s’est bien passée.**  false**: erreur SQL** |

|  |
| --- |
| display |
| Fourni un affichage simplifié du résultat des méthodes qui renvoient un tableau.  **Description**  **display**(string $libelle, array $tableau)  **Liste des paramètres**  **libelle**  **Libellé libre à afficher en entête.**  **tableau**  Tableau de résultat suite à l’appel d’une méthode de récupération d’arborescence.  **Retour**  true**: l’insertion s’est bien passée.**  false**: erreur SQL**  **Exemple**  //affichage de l'arbre complet créé  $res = $tree->**getTree**(0, false, false);  $tree->**display**('ARBRE COMPLET HIERARCHIQUE', $res);  Renvoie  ARBRE COMPLET HIERARCHIQUE  1 (1)  … 2 (2)  …… 4 (4)  …… 5 (5)  ……… 9 (9)  ………… 11 (11)  ………… 12 (12)  ………… 13 (13)  ………… 14 (14)  ……… 10 (10)  ………… 15 (15)  ………… 16 (16)  ………… 17 (17)  …… 6 (6)  …… 7 (7)  …… 8 (8)  … 3 (3)  C’est une méthode basique dites de débogage car elle ne permet que d’afficher les feuilles et leur identifiant. Il faudra la surcharger dans votre classe héritée pour un meilleur rendu. |

## Références

<https://openclassrooms.com/courses/la-representation-intervallaire>  
<http://sqlpro.developpez.com/cours/arborescence/>