

La représentation intervallaire

Par Baptiste Clavié (Talus)



www.openclassrooms.com

*Licence Creative Commons 7 2.0
Dernière mise à jour le 10/09/2010*

Sommaire

Sommaire	2
Lire aussi	1
La représentation intervallaire	3
La théorie	3
Une (petite) présentation de tout le "schmilblik"	3
Pourquoi préférer la RI à l'héritage récursif ? Les tables SQL d'exemple	5
La pratique	8
Stats sur les différents noeuds	8
Mettre à jour un élément	11
Partager	19

La représentation intervallaire



Par

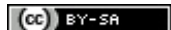
Baptiste Clavié (Talus)

Mise à jour : 10/09/2010

Difficulté : Intermédiaire



Durée d'étude : 10 heures



Une notion bien étrange que voilà...

Pour mieux expliquer, prenons (par exemple) des forums, ayant plusieurs "niveaux" (comprenez sous-forums). Pour accéder aux "enfants" et aux "parents" d'un forum, vous pensez sûrement (et, je pense, vous les utiliserez) les héritages (présence d'une colonne `parent_id`, méthode également connue sous le nom de *représentation classique*) par auto-jointures, c'est-à-dire en renseignant récursivement la clause `WHERE` à l'aide de sous-requêtes.

Mais, ceci peut **très** vite devenir lourd... et peu pratique, surtout à partir d'un certain niveau (et qui, au grand dam de tous, est vite atteint).



Notez que, tout au long du tutoriel, pour la plupart des exemples, j'utiliserai l'exemple de forums.



Dans ce tuto, nous allons donc voir comment éviter cette lourdeur en employant une "ruse" mise au point par les développeurs : **la représentation intervallaire** (nommée RI pour les intimes) !

Accrochez-vous, on y va !

Sommaire du tutoriel :



- [La théorie](#)
- [La pratique](#)

La théorie

Une (petite) présentation de tout le "schmilblik"

Avant d'attaquer la bête, nous avons, tout d'abord, besoin de la définir. Tout d'abord, qu'est-ce que la repr...



C'est vrai ça, c'est quoi la rapré... représentation invert... cette chose ?

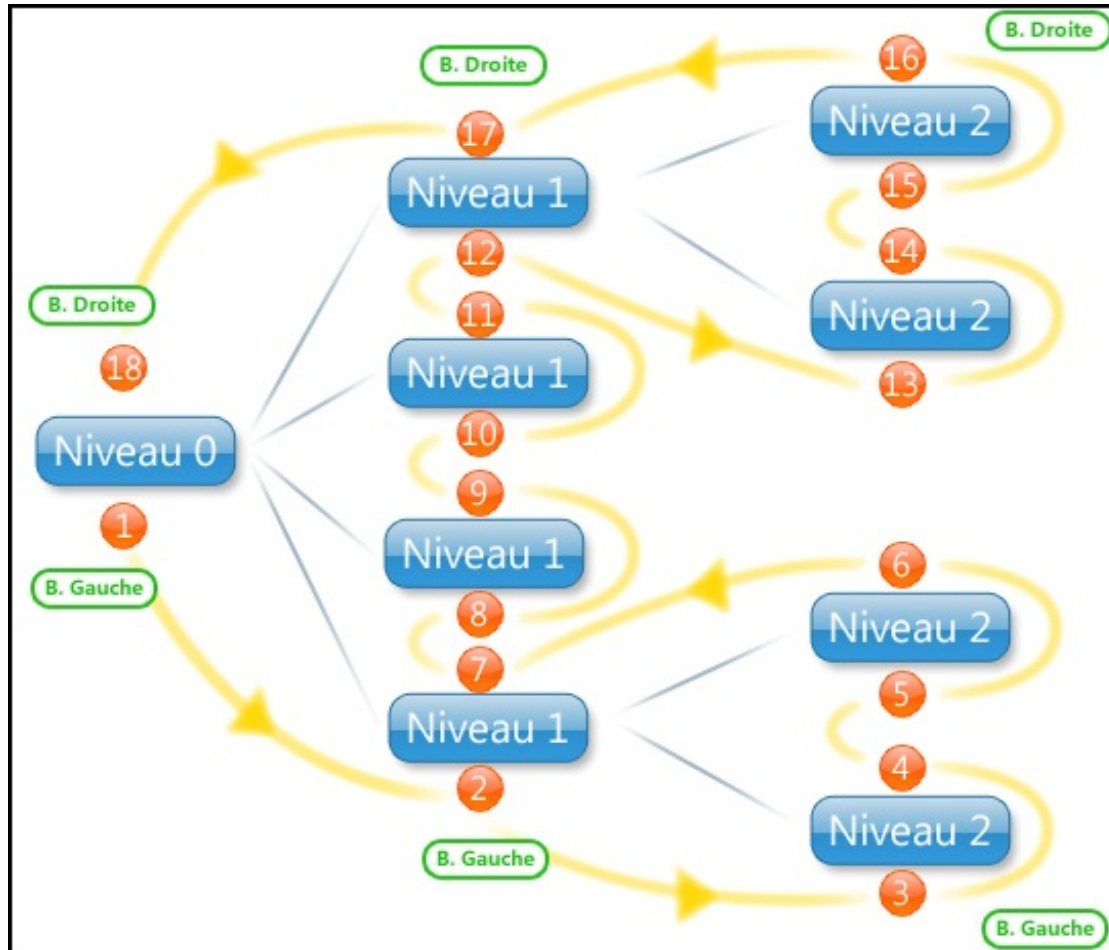
La représentation intervallaire ! Je la voyais arriver à des kilomètres à la ronde, celle-là !

Je commence par un point optionnel (ce sera plus facile à expliquer). Cette technique peut permettre de situer un élément dans une hiérarchie. Pour donner une image de cette notion de hiérarchie, prenons l'exemple d'un immeuble, couplé à un système de forums (vous allez très vite comprendre). Vous savez, un immeuble est constitué d'étages...

Disons que la catégorie d'un système de forums est le rez-de-chaussée de cet immeuble, et les étages, les sous-forums (qui sont chacun des sous-forums de l'étage précédent). Voilà ce qu'est la notion de hiérarchie : si je suis au quatrième étage (non, non, je ne vais pas sauter 😊), ça veut dire que je suis... au quatrième niveau, ou à la quatrième hiérarchie. Il peut, bien entendu, y avoir plusieurs sous-forums ayant le même niveau, comme, toujours à l'image d'un immeuble, il peut y avoir plusieurs appartements à chaque étage.

Ensuite, un point **essentiel** à *retenir* dans la représentation intervallaire (je ne vois pas ce qu'il y a de compliqué à retenir là-dedans, ce n'est quand même pas de l'acide acétyli... acide acétyli... bref, un certain médicament 🤪), les notions de *borne gauche*, *borne droite*.

Pour illustrer cette notion, voici un schéma fait par mes soins (mes capacités en graphisme étant très limitées, vous m'excuserez de la qualité de celui-ci), qui, je dois vous l'avouer, reprend un peu la même idée que [le schéma de Frédéric Brouard](#) (le mien étant simplifié, et (surtout) plus adapté à notre cas de développeurs en herbe), qui d'ailleurs présente de très bons tutos sur SQL.



Merci à [Shadow_f](#) pour le schéma

Comme vous pouvez le voir, chaque borne de chaque élément se voit attribuer un nombre. Ce nombre permet de déterminer la position de l'élément dans l'ensemble. Notez également que *le nombre de la borne gauche est toujours inférieur au nombre de la borne droite*. Notez bien ce point, il est très important (je fais une liste de rappels à la fin de cette première sous-partie) !

À propos de points importants, en voilà un autre à noter : si **la différence entre sa borne droite et sa borne gauche** (et non pas l'inverse !) **est égale à 1**, on désigne alors l'élément étant *une feuille*. Sinon, ce sera un *noeud*, et cette même différence sera égale au **double du nombre d'enfants**, auquel on ajoute 1.

Je pense que vous constaterez qu'au lieu de mettre des noms aux éléments sur mon schéma, j'ai juste indiqué les niveaux de chaque élément, pour vous montrer la tête que ça peut avoir.



Euh... C'est bien beau tout ça, mais quels sont les points essentiels de la représentation intervallaire ?

J'y viens, et vous posez la question pile au bon moment. Je vous ai fait une petite liste de trucs à retenir...

- La notion de niveaux détermine la hiérarchisation de la feuille, qui, je dois vous l'avouer, n'est pas essentielle à la RI, mais plutôt utile (pour les tris par exemple).
- La notion de bornes, qui détermine la situation de la feuille (ou du noeud) par rapport à l'ensemble.
- Une feuille désigne un élément caractérisé par le fait que la différence entre sa borne droite et sa borne gauche est égale à 1.

1. Un noeud, lui, est un élément caractérisé par les autres cas (donc la différence borne droite - borne gauche supérieure à 1).
- Pour calculer la différence entre les deux bornes, il faut **TOUJOURS** soustraire la borne gauche à la borne droite, et non pas l'inverse. En effet, la borne gauche est **TOUJOURS inférieure** à la borne droite.
 - La différence entre les deux bornes d'un même noeud (et non pas d'une feuille) est (quoi qu'il advienne) **TOUJOURS** égale au *double* du nombre d'enfants, auquel on rajoute 1.

Passons maintenant à la préparation pour pouvoir pratiquer, dans la seconde partie de ce tutoriel (majoritairement des requêtes SQL, pour servir à titre d'exemples). Je montrerai également pourquoi il est difficile (et coûteux) d'utiliser les "auto-jointures" (c'est-à-dire d'utiliser avec abus la clause WHERE et les sous-requêtes) pour accéder aux noeuds parents et aux feuilles... Et que c'est même presque impossible pour nous en ce moment. À moins d'avoir des idées tordues... 😊

Pourquoi préférer la RI à l'héritage récursif ? Les tables SQL d'exemple



Dans cette sous-partie, je considère que vous savez manipuler **SELECT** et ses clauses **WHERE**, **ORDER BY**, et (éventuellement) comprendre les **requêtes d'insertion**. Sinon, vous avez la porte, le big tutoriel de M@teo21, "Faire un site dynamique avec PHP - Base de Données", le big tutoriel de Shepard, "Pour aller plus loin..." à disposition.

L'héritage récursif ? Ourf, c'est lourd... et infaisable (pour des Zéros comme nous) 😊

Voici la table type que nous allons utiliser pour illustrer la faiblesse (😬) de l'héritage récursif :

Code : SQL

```
CREATE TABLE tuto_herit (
    forum_id mediumint(8) NOT NULL AUTO_INCREMENT,
    forum_parent_id mediumint(8) DEFAULT NULL,
    forum_name varchar(30) NOT NULL,

    PRIMARY KEY (forum_id),
    KEY forum_parent_id (forum_parent_id)
);

INSERT INTO tuto_herit (forum_id, forum_parent_id, forum_name)
VALUES
    (1, NULL, '1'),
    (2, 1, '1.1'),
    (3, 2, '1.1.1'),
    (4, 3, '1.1.1.1'),
    (5, 4, '1.1.1.1.1'),
    (6, 5, '1.1.1.1.1.1'),
    (7, 6, '1.1.1.1.1.1.1'),
    (8, 6, '1.1.1.1.1.1.2'),
    (9, 6, '1.1.1.1.1.1.3'),
    (10, 4, '1.1.1.1.1.2'),
    (11, 4, '1.1.1.1.1.3'),
    (12, 4, '1.1.1.1.2'),
    (13, 4, '1.1.1.1.3'),
    (14, 2, '1.1.2'),
    (15, 1, '1.2'),
    (16, 1, '1.3'),
    (17, 1, '1.4'),
    (18, 11, '1.4.1'),
    (19, 11, '1.4.2'),

    (20, NULL, '2') /*, [...] */;
```



Notez que j'ai délibérément choisi de nommer les éléments en fonction de leur parent, pour pouvoir les distinguer. Ainsi, X.YZ veut dire que l'élément correspondant appartient à un parent X, qui a un fils Y, ... etc. Mais vous êtes libres de nommer les tables comme vous le souhaitez, ce nommage n'étant **pas** une condition pour l'utilisation de la RI.

À partir d'un des sous-forums (on va dire celui ayant l'id numéro 7, le plus profond 🤖), comment faire, à l'aide des jointures, pour accéder à la catégorie 1, en affichant tous les forums parents au sous-forum sélectionné ? Je vous laisse 5 minutes pour essayer de faire cette requête... Et bien entendu, en ayant qu'un seul forum par ligne (pas tous, sinon ce serait trop facile, surtout si on connaît la profondeur du sous-forum 🤖).

...

...

Vous n'y arrivez pas ? Pour tout vous avouer, moi non plus 🤖. Ici, on peut facilement avoir tous les forums parents sur la **même ligne** à l'aide de *quelques* jointures (en utilisant l'héritage de la colonne forum_parent_id) ; or, ce n'était pas ce qui était demandé : on voulait un forum par ligne. Certes, les jointures, c'est pratique, mais à la longue... c'est vite pesant et peu pratique.



N.d.A. : pour ce problème, je pense qu'il faut soit utiliser la récursivité d'une fonction pour trouver ce qu'on cherche, mais ça reste consommateur de ressource si l'arbre est trop profond (dans notre cas, cela peut être considéré comme trop profond), soit, toujours dans la récursivité, une série de sous-requêtes, ce qui n'est également pas terrible.

La représentation intervallaire ? Yeah !

Reprenons la première table... En l'adaptant quelque peu pour pouvoir la rendre utilisable par la représentation intervallaire...

Hop, la voilà prête à l'emploi (toujours avec les insertions, converties qui plus est 🤖) !

Code : SQL

```
CREATE TABLE tuto_ri (
    forum_id mediumint(8) NOT NULL AUTO_INCREMENT,
    forum_level mediumint(8) NOT NULL DEFAULT 0,
    forum_gauche mediumint(8) NOT NULL,
    forum_droite mediumint(8) NOT NULL,
    forum_name varchar(30) NOT NULL,

    PRIMARY KEY (forum_id),
    KEY forum_gauche (forum_gauche),
    KEY forum_droite (forum_droite)
);

INSERT INTO tuto_ri (forum_id, forum_level, forum_gauche,
forum_droite, forum_name)
VALUES
    (1, 0, 1, 38, '1'),
    (2, 1, 2, 27, '1.1'),
    (3, 2, 3, 24, '1.1.1'),
    (4, 3, 4, 23, '1.1.1.1'),
    (5, 4, 5, 18, '1.1.1.1.1'),
    (6, 5, 6, 13, '1.1.1.1.1.1'),
    (7, 6, 7, 8, '1.1.1.1.1.1.1'),
    (8, 6, 9, 10, '1.1.1.1.1.1.2'),
    (9, 6, 11, 12, '1.1.1.1.1.1.3'),
    (10, 5, 14, 15, '1.1.1.1.1.2'),
    (11, 5, 16, 17, '1.1.1.1.1.3'),
    (12, 4, 19, 20, '1.1.1.1.2'),
    (13, 4, 21, 22, '1.1.1.1.3'),
    (14, 2, 25, 26, '1.1.2'),
    (15, 1, 28, 29, '1.2'),
    (16, 1, 30, 31, '1.3'),
    (17, 1, 32, 37, '1.4'),
    (18, 2, 33, 34, '1.4.1'),
```

```
(19, 2, 35, 36, '1.4.2'),
(20, 0, 39, 40, '2') /*, [...] */;
```

La note concernant les noms des (sous-)forums reste toujours valable dans ce cas-ci.

Allez : même exercice que tout à l'heure ! Vous avez... 10 minutes pour le faire : rechercher l'id, le nom, et la hiérarchie des noeuds parents de la feuille ayant l'id "7", en les triant du parent le plus proche au parent le plus lointain !

...



...

...

...

...

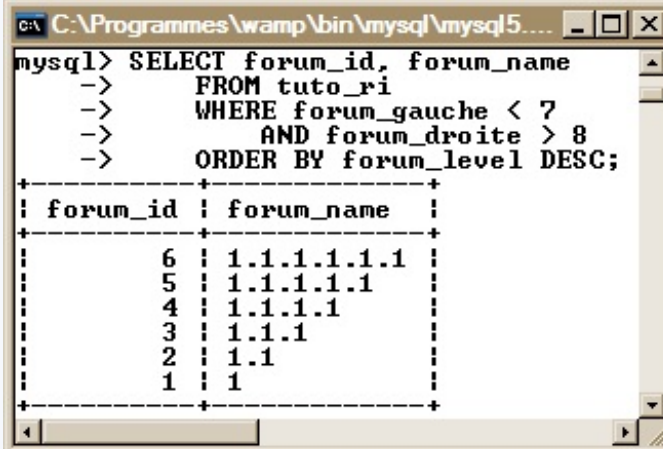
...

Ah, c'est vrai, j'avais oublié le petit indice : utiliser les bornes des feuilles. Si vous n'aviez pas deviné (nan, surtout pas, pas la fenêtre !), ce n'est pas grave, voici ma solution, et pour tenter de me faire pardonner () , je l'ai commentée pour que vous la compreniez mieux).

Secret ([cliquez pour afficher](#))

Code : SQL

```
SELECT forum_id, forum_name
FROM tuto_ri
WHERE forum_gauche < 7
      AND forum_droite > 8
ORDER BY forum_level DESC;
```




```
mysql> SELECT forum_id, forum_name
->      FROM tuto_ri
->      WHERE forum_gauche < 7
->            AND forum_droite > 8
->            ORDER BY forum_level DESC;
```

forum_id	forum_name
6	1.1.1.1.1.1
5	1.1.1.1.1
4	1.1.1.1
3	1.1.1
2	1.1
1	1

On retrouve donc ce qu'on recherchait

On cherche à récupérer l'ID, le nom, et la hiérarchisation des noeuds parents, ce qu'on fait en sélectionnant les champs `forum_id`, `forum_name`, et `forum_level`.

Ensuite, dans la clause `WHERE`, on applique ce qu'on a pu observer sur mon (beau ) schéma : la borne gauche d'un des

(sous-)forums est *toujours* **plus grande** que celle des noeuds parents. On sélectionne donc tous les noeuds ayant une borne gauche plus petite que (dans notre cas) 7. On applique la même chose à la borne droite, sauf que cette fois-ci, la borne droite de la feuille est *toujours* **plus petite** que celle des noeuds parents.

On a donc les noeuds que l'on désire... Mais ce n'est pas fini, il y avait un petit piège : on souhaite avoir les noeuds... Du plus proche au plus lointain par rapport à la feuille. Il suffisait juste d'ajouter une petite clause ORDER BY, en indiquant le nom de la colonne qui sert à trier (forum_level), et par ordre **décroissant**. Je vous avais bien dit que la notion de hiérarchisation pouvait s'avérer utile par moments... En voilà la preuve même !



Notez que si je vous avais demandé, dans la liste des forums ressortie, de ressortir également le forum dans lequel on est, à la place des opérateurs < et >, il aurait fallu utiliser <= et >=. En effet, les bornes gauche et droite du forum sont bien plus petites ou égales aux bornes de notre forum 😊

```

mysql> SELECT forum_id, forum_name
->      FROM tuto_ri
->      WHERE forum_gauche <= 7
->            AND forum_droite >= 8
->            ORDER BY forum_level DESC;
+-----+-----+
| forum_id | forum_name |
+-----+-----+
| 7       | 1.1.1.1.1.1.1 |
| 6       | 1.1.1.1.1.1 |
| 5       | 1.1.1.1.1 |
| 4       | 1.1.1.1 |
| 3       | 1.1.1 |
| 2       | 1.1 |
| 1       | 1 |
+-----+-----+
7 rows in set (0.00 sec)

```

Ici, on cherche aussi à inclure le forum courant dans le résultat

Bon, je vous pense prêts pour la pratique : à l'assauuuuuut 🧑🏻 !

La pratique

Ici, nous allons voir les différentes techniques et opérations ~~chirurgicales~~ liées à la représentation intervallaire : modifier le statut d'un élément (transformer un noeud en une feuille, et vice-versa), ajouter des feuilles à un noeud, compter le nombre d'éléments dans un noeud, pouvoir mettre à jour les indices aux bornes en fonction du nombre d'éléments (selon que ce sera une feuille ou un noeud), le déplacement de noeuds...

Cette partie du tuto agira donc en tant que plusieurs séries de "minis-TP", qui demandent par moments de la réflexion, et par moments des astuces... que je développerai ici.

Stats sur les différents noeuds

Dans cette sous-partie, nous allons voir comment compter le nombre d'éléments d'un noeud, le nombre de feuilles d'un noeud, le nombre de noeuds dans un même noeud, ... bref, on va compter. 🤖

Compter le nombre d'éléments d'un noeud

Pour compter le nombre d'éléments d'un noeud, ça va être relativement (très) simple, et (très) vite expédié. Vous vous souvenez, à la fin de la première partie de ce tuto, on a vu comment récupérer tous les parents (et leur caractéristiques) d'une feuille / d'un noeud, en incluant ou non l'élément, non ? Eh bien ici, ça va plus ou moins être le même topo.




Ha ! Laisse-moi deviner : au lieu d'inclure l'élément dans la sélection, on va déjà devoir utiliser les signes strictement



supérieur / inférieur, non ?


Bingo ! De plus, vous aurez besoin, comme je l'ai dit, **non pas** de sélectionner *les parents* de l'élément, **mais ses enfants** ! Donc, au lieu de rechercher tous les éléments qui ont une borne gauche inférieure à celle de l'élément recherché, et une borne droite supérieure à celle de l'élément, on va faire l'inverse !

Je vous file la solution dans quelques minutes, le temps pour vous de chercher (et moi d'aller piquer un petit somme ). Ah, j'oubliais pour quel forum on va faire cette opération : cette fois-ci, on va chercher le nombre d'enfants pour le forum ayant l'ID numéro... 3 !

Petit indice : utilisez la fonction d'agrégat COUNT(*) (pour les fonctions d'agrégat, voyez [le tutorial de Shepard](#) sur le sujet).

...

...

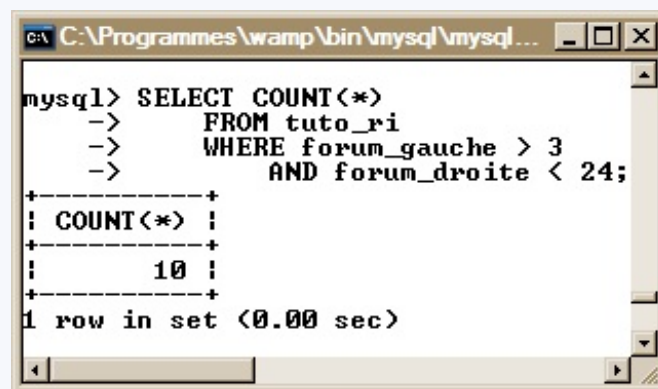
Hmm ? Vous avez (déjà) fini ? Voici donc la correction .

Secret ([cliquez pour afficher](#))

Code : SQL

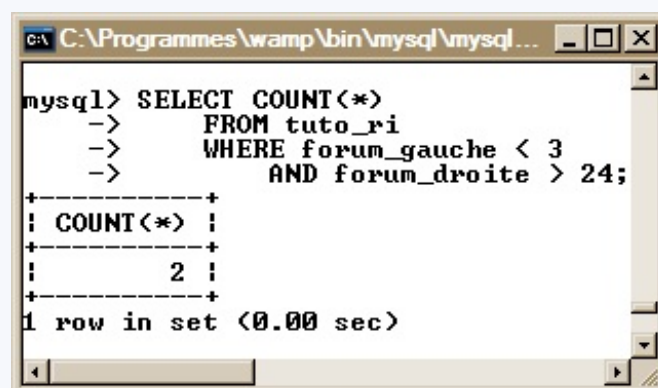
```
SELECT COUNT(*)
  FROM tuto_ri
 WHERE forum_gauche > 3
        AND forum_droite < 24;
```

Vous avez vu, ce n'était pas si sorcier, il suffisait donc juste d'inverser les signes pour les deux bornes pour obtenir les enfants et non pas les parents du noeud, et d'utiliser, comme je vous l'ai dit, la fonction d'agrégat COUNT(*).



```
mysql> SELECT COUNT(*)
->      FROM tuto_ri
->      WHERE forum_gauche > 3
->            AND forum_droite < 24;
+-----+
| COUNT(*) |
+-----+
|        10 |
+-----+
1 row in set (0.00 sec)
```

On trouve donc 10 éléments fils au noeud '1.1.1'



```
mysql> SELECT COUNT(*)
->      FROM tuto_ri
->      WHERE forum_gauche < 3
->            AND forum_droite > 24;
+-----+
| COUNT(*) |
+-----+
|         2 |
+-----+
1 row in set (0.00 sec)
```

Dans le sens inverse, on retrouve 2 éléments parents au noeud '1.1.1'



Vous verrez par la suite (bon, tout de suite après, pour tout vous avouer 😊) que le nombre d'éléments parents est forcément celui des noeuds parents (c'est de la pure logique 😊).

Compter le nombre de feuilles / de noeuds d'un noeud



Mais... Mais... N'est-ce pas la même chose que ce qu'on a fait à l'instant même ?

Pas tout à fait. Ici, on ne cherchera pas à compter tous les **enfants** (ou les **parents**) d'un noeud, mais **seulement les feuilles (ou les noeuds) enfants (ou les parents) d'un noeud**. Donc, on va à peu près procéder de la même manière qu'auparavant, mais, tâchez juste de vous rappeler d'un des points à retenir de la représentation intervallaire. Vous y êtes ? Non ? Bon bah cherchez le nombre de noeuds enfants pour le même forum que le précédent 😊 (et ça va me permettre d'aller chercher un café pour finir de me réveiller ☹).

...

...

...

...

Que vous ayez fini ou non, voici la correction !

Secret ([cliquez pour afficher](#))

Code : SQL

```
SELECT COUNT(*)
FROM tuto_ri
WHERE forum_gauche > 3
      AND forum_droite < 24
      AND (forum_droite - forum_gauche) = 1;
```

```
C:\Programmes\wamp\bin\mysql\mysql5.0.51b\bin\mysql.exe

mysql> SELECT COUNT(*)
      FROM tuto_ri
      WHERE forum_gauche > 3
      AND forum_droite < 24
      AND (forum_droite - forum_gauche) = 1;
+-----+
| COUNT(*) |
+-----+
|          7 |
+-----+
1 row in set (0.00 sec)
```

On trouve 7 feuilles filles pour le "Forum 1.1.1"

Ce TP était également plutôt facile ; il suffisait de se souvenir que pour toutes les feuilles d'un arbre, la différence entre les deux bornes est toujours égale à 1. De même, pour compter le nombre de noeuds fils, vous pouvez, je pense, aisément le deviner : il faut chercher non pas les fils ayant une différence entre leurs bornes égale à 1, mais supérieure à 1.

```
C:\Programmes\wamp\bin\mysql\mysql5.0.51b\bin\mysql.exe

mysql> SELECT COUNT(*)
-> FROM tuto_ri
-> WHERE forum_gauche > 3
-> AND forum_droite < 24
-> AND (forum_droite - forum_gauche) > 1;
+-----+
| COUNT(*) |
+-----+
|         3 |
+-----+
1 row in set (0.00 sec)
```

On trouve 3 noeuds fils pour le noeud "1.1.1"

```
C:\Programmes\wamp\bin\mysql\mysql5.0.51b\bin\mysql.exe

mysql> SELECT COUNT(*)
-> FROM tuto_ri
-> WHERE forum_gauche < 3
-> AND forum_droite < 24
-> AND (forum_droite - forum_gauche) = 1;
+-----+
| COUNT(*) |
+-----+
|         0 |
+-----+
1 row in set (0.00 sec)
```

On trouve 0 feuilles parentes pour le noeud "1.1.1"

```
C:\Programmes\wamp\bin\mysql\mysql5.0.51b\bin\mysql.exe

mysql> SELECT COUNT(*)
-> FROM tuto_ri
-> WHERE forum_gauche < 3
-> AND forum_droite > 24
-> AND (forum_droite - forum_gauche) > 1;
+-----+
| COUNT(*) |
+-----+
|         2 |
+-----+
1 row in set (0.00 sec)
```

On trouve 2 noeuds parents pour le noeud "1.1.1"



Pour le troisième résultat, je suppose que vous vous en doutiez : un enfant ne peut avoir de feuilles parentes. Ou alors c'est qu'il y a un problème au niveau des bornes. 🤔



De même, vous vous doutez que **si vous mettez un signe supérieur (respectivement inférieur) ou égal à 1 pour la différence**, ça équivaut à la même chose que tout à l'heure... Si ce n'est de pomper des ressources inutilement. 😊



Dernière remarque : si vous additionnez les deux premiers résultats, vous pourrez retrouver le résultat qu'on a vu plus haut, soit tous les éléments fils du noeud "Forum 1.1.1"... Même remarque pour les éléments parents. 😊

Mettre à jour un élément

Ici, on va voir comment mettre à jour un élément : mettre ses bornes à jour (ajout / suppression d'éléments), mise à jour qui peut entraîner la transformation d'une feuille en un noeud (et vice-versa 🤪)... Bref, on va s'amuser à mettre à jour notre arbre. 😊

Ajout / suppression d'éléments (transformation d'une feuille en un noeud "simple", d'un noeud "simple" en une feuille)

Déjà, vous devez savoir comment insérer / supprimer des lignes d'une base de données, à l'aide des requêtes **INSERT** et **DELETE**... Vous y êtes ? On va insérer une nouvelle ligne par la droite à la feuille de niveau 4, 1.[...].3 (ID N°13). Pour cela, puisque nous connaissons déjà la borne gauche, la borne droite, et le niveau de la feuille, on évite une requête. Enfin, c'est à vous de voir si vous voulez (ou non) ajouter une requête **SELECT**, ça dépend des cas.

De plus, on aura également besoin de faire 2 *updates* : changer les bornes gauches et droites de tous les éléments à partir desquels on souhaite insérer la nouvelle feuille. Donc, à chacune des bornes, on va incrémenter tout d'abord la borne droite de 2, puis la borne gauche de... 2 également, et le tout en 2 requêtes. Pourquoi deux ? Pour ne pas nous embêter à pondre un truc trop complexe, et pour une raison que je préciserai dans le corrigé, plus bas. Après tout, pourquoi faire complexe quand on peut faire simple ?

Et enfin, on insère la nouvelle feuille... par la droite. Pourquoi par la droite ? Parce que c'est plus pratique... 🤪

Mais après, libre à vous de suivre le conseil que je vous donne (qui est également celui de M Brouard). 😊

Ici, on cherche donc à insérer notre nouvelle feuille par la droite : je vous laisse un peu de temps (pour percuter) et essayer de pondre les requêtes.



Euh... Mais attends, je comprends pas, pourquoi appelles-tu ceci un noeud "simple" 🤪 ?

Ah, c'est vrai, je n'ai pas détaillé. J'appelle ici noeud "simple" le fait que ce noeud n'ait qu'une feuille, et le fait que l'on insère que des feuilles.... et non pas des noeuds. Je détaillerai dans le prochain mini-TP comment procéder dans ce cas (pour retirer un noeud d'un arbre aussi 🤪). Bref, allez-y, la correction arrive dans quelques instants 😊.

...

...

...

Voici la correction tant désirée. 😊

Secret (cliquez pour afficher)

Code : SQL

```
UPDATE tuto_ri
  SET forum_droite = forum_droite + 2
  WHERE forum_droite >= 22;

UPDATE tuto_ri
  SET forum_gauche = forum_gauche + 2
  WHERE forum_gauche >= 22;

INSERT INTO tuto_ri (forum_level, forum_gauche, forum_droite,
forum_name)
  VALUES (5, 22, 23, '1.1.1.1.3.1');
```

```

mysql> UPDATE tuto_ri
->     SET forum_droite = forum_droite + 2
->     WHERE forum_droite >= 22;
Query OK, 12 rows affected (0.00 sec)
Rows matched: 12  Changed: 12  Warnings: 0

mysql>
mysql> UPDATE tuto_ri
->     SET forum_gauche = forum_gauche + 2
->     WHERE forum_gauche >= 22;
Query OK, 7 rows affected (0.00 sec)
Rows matched: 7  Changed: 7  Warnings: 0

mysql>
mysql> INSERT INTO tuto_ri (forum_level, forum_gauche
->     VALUES (5, 22, 23, '1.1.1.1.3.1');
Query OK, 1 row affected (0.00 sec)

```

Insertion d'une feuille dans un noeud / une feuille

Vous ayant tout expliqué avant, c'était juste une application de ce que j'ai dit plus haut. 😊

Lorsqu'on sélectionne les données, on trouve la feuille insérée à la bonne place (ici, je n'ai sélectionné que les colonnes forum_gauche, forum_droite, et forum_name, pour éviter d'avoir une largeur trop grande pour mon screen) :

```

mysql> SELECT forum_name, forum_gauche, forum_droite
->     FROM tuto_ri
->     ORDER BY forum_gauche ASC;

```

forum_name	forum_gauche	forum_droite
1	1	40
1.1	2	29
1.1.1	3	26
1.1.1.1	4	25
1.1.1.1.1	5	18
1.1.1.1.1.1	6	13
1.1.1.1.1.1.1	7	8
1.1.1.1.1.1.2	9	10
1.1.1.1.1.1.3	11	12
1.1.1.1.1.2	14	15
1.1.1.1.1.3	16	17
1.1.1.1.2	19	20
1.1.1.1.3	21	24
1.1.1.1.3.1	22	23
1.1.2	27	28
1.2	30	31
1.3	32	33
1.4	34	39
1.4.1	35	36
1.4.2	37	38
2	41	42

21 rows in set (0.01 sec)

OK, tout est correct !



Vous pouvez ainsi transformer une "simple" feuille en un noeud... "simple".

Je pense que vous devinerez comment supprimer une feuille :

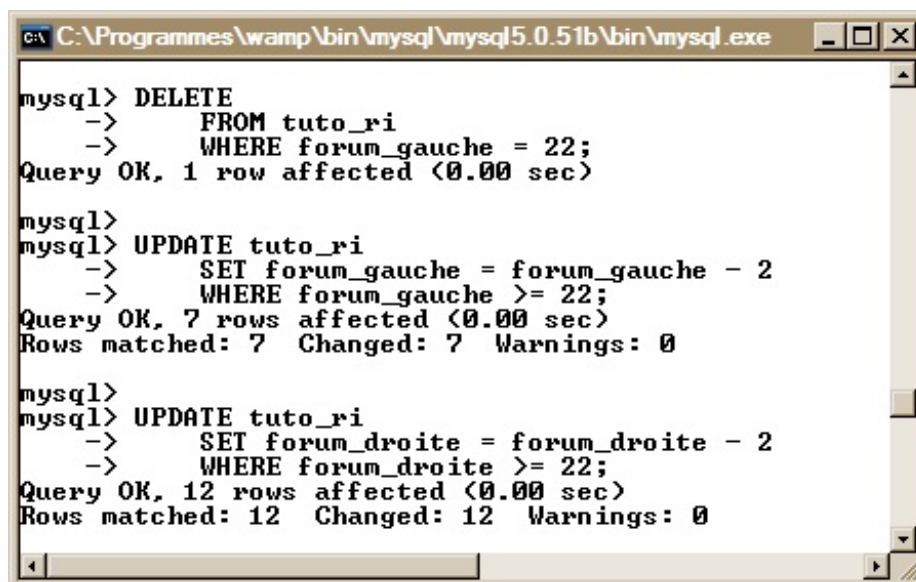
Code : SQL

```
DELETE
  FROM tuto_ri
  WHERE forum_gauche = 22;

UPDATE tuto_ri
  SET forum_gauche = forum_gauche - 2
  WHERE forum_gauche >= 22;

UPDATE tuto_ri
  SET forum_droite = forum_droite - 2
  WHERE forum_droite >= 22;
```

Le topo est à peu près le même que pour l'insertion, sauf que cette fois-ci, on procède d'abord à la suppression, puis à la mise à jour de la borne gauche, et enfin à la mise à jour de la borne droite ; j'expliquerai pourquoi, lors de l'insertion, on commence toujours par la mise à jour de la borne droite, puis on s'occupe de la borne gauche, puis de l'insertion, et dans le sens inverse lors de la suppression (enfin, on change l'insertion par la suppression, je pense que pour ça, vous avez déjà percuté 🤔), plus bas pour résumer ce TP.



```
C:\Programmes\wamp\bin\mysql\mysql5.0.51b\bin\mysql.exe

mysql> DELETE
->     FROM tuto_ri
->     WHERE forum_gauche = 22;
Query OK, 1 row affected (0.00 sec)

mysql>
mysql> UPDATE tuto_ri
->     SET forum_gauche = forum_gauche - 2
->     WHERE forum_gauche >= 22;
Query OK, 7 rows affected (0.00 sec)
Rows matched: 7  Changed: 7  Warnings: 0

mysql>
mysql> UPDATE tuto_ri
->     SET forum_droite = forum_droite - 2
->     WHERE forum_droite >= 22;
Query OK, 12 rows affected (0.00 sec)
Rows matched: 12  Changed: 12  Warnings: 0
```

Suppression d'une feuille dans un noeud / retransformation d'un noeud "simple" en une feuille

```

mysql> SELECT forum_name, forum_gauche, forum_droite
-> FROM tuto_ri
-> ORDER BY forum_gauche ASC;
+-----+-----+-----+
| forum_name | forum_gauche | forum_droite |
+-----+-----+-----+
| 1          | 1            | 38           |
| 1.1        | 2            | 27           |
| 1.1.1      | 3            | 24           |
| 1.1.1.1    | 4            | 23           |
| 1.1.1.1.1  | 5            | 18           |
| 1.1.1.1.1.1 | 6           | 13           |
| 1.1.1.1.1.1.1 | 7          | 8            |
| 1.1.1.1.1.1.1.2 | 9         | 10           |
| 1.1.1.1.1.1.1.3 | 11        | 12           |
| 1.1.1.1.1.2 | 14          | 15           |
| 1.1.1.1.1.3 | 16          | 17           |
| 1.1.1.1.2   | 19          | 20           |
| 1.1.1.1.3   | 21          | 22           |
| 1.1.1.2     | 25          | 26           |
| 1.2         | 28          | 29           |
| 1.3         | 30          | 31           |
| 1.4         | 32          | 37           |
| 1.4.1       | 33          | 34           |
| 1.4.2       | 35          | 36           |
| 2          | 39          | 40           |
+-----+-----+-----+
20 rows in set (0.00 sec)

```

On retourne à l'état initial



Alors ? Pourquoi recommandes-tu d'exécuter ces actions dans cet ordre précis ?

C'est juste pour ne pas avoir d'erreurs trop bêtes, si vous avez mis des contraintes d'unicité sur vos deux bornes. Ainsi, lors de l'insertion, on décale d'abord toutes les bornes à partir desquelles on veut insérer notre feuille, pour pouvoir ensuite décaler les bornes gauches des feuilles suivantes sans que la borne gauche ne soit jamais égale à une borne droite, et vice-versa (pewh). Ensuite seulement, on insère la feuille à la borne désirée. Pour la suppression, c'est à peu près le même topo, mais dans le sens inverse.



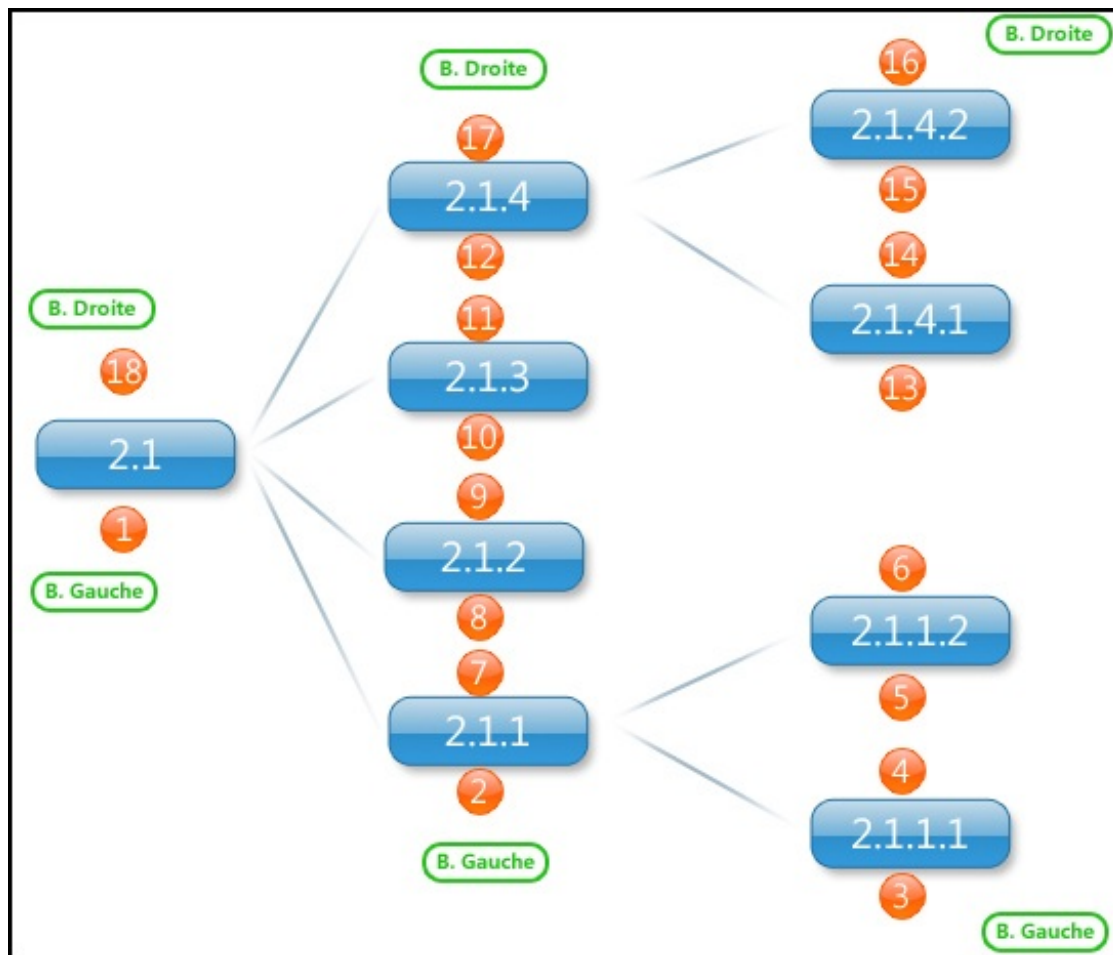
Pour la suppression, vous n'êtes pas obligés de tout re-décaler, mais suivez mon conseil, et faites-le, histoire d'avoir un arbre plus propre et ne pas avoir "trop" de trous ~~et pas mal d'emmerdes~~ au fur et à mesure des différentes opérations.

Maintenant, si vous êtes toujours là, on va aborder la même chose... Mais avec des noeuds, c'est-à-dire l'insertion et la suppression d'un noeud dans un arbre ! 😊

Insertion / suppression d'un sous-arbre (noeud) dans un arbre

Je ne pense pas avoir besoin de trop détailler. Vous avez juste besoin de connaître les bornes de l'arbre à insérer, le nombre d'éléments qu'il contient, et ça devrait faire l'affaire, si on emploie la même méthode que tout à l'heure. 😊

Voici l'arbre que l'on va chercher à insérer :



Merci à [Shadow_f](#) pour le schéma

On va donc rattacher cet arbre à la feuille "2", c'est-à-dire celle étant comprise entre les bornes... 39 et 40. En ayant cet arbre, et sachant qu'il y a **9 éléments**, la différence entre les deux bornes de la catégorie 2 va être égale à (je sors ma calculatrice 😊) : $2 * 9 + 1$, soit 19. La borne droite de notre nouveau "conteneur" sera donc de 58 (car, d'après ma calculatrice, $58 - 19 = \dots 39$!); on aura aussi besoin d'incrémenter toutes les bornes droites du nœud à insérer du double du nombre d'éléments, soit... $9 * 2.. 18$!

Je vous laisse cogiter un peu (n'oubliez pas de décaler les bornes de l'arbre à insérer... Vous verrez que c'est plutôt facile 😊)...

...

...

Rendez vos copies, voilà la correction !

Secret ([cliquez pour afficher](#))

Code : SQL

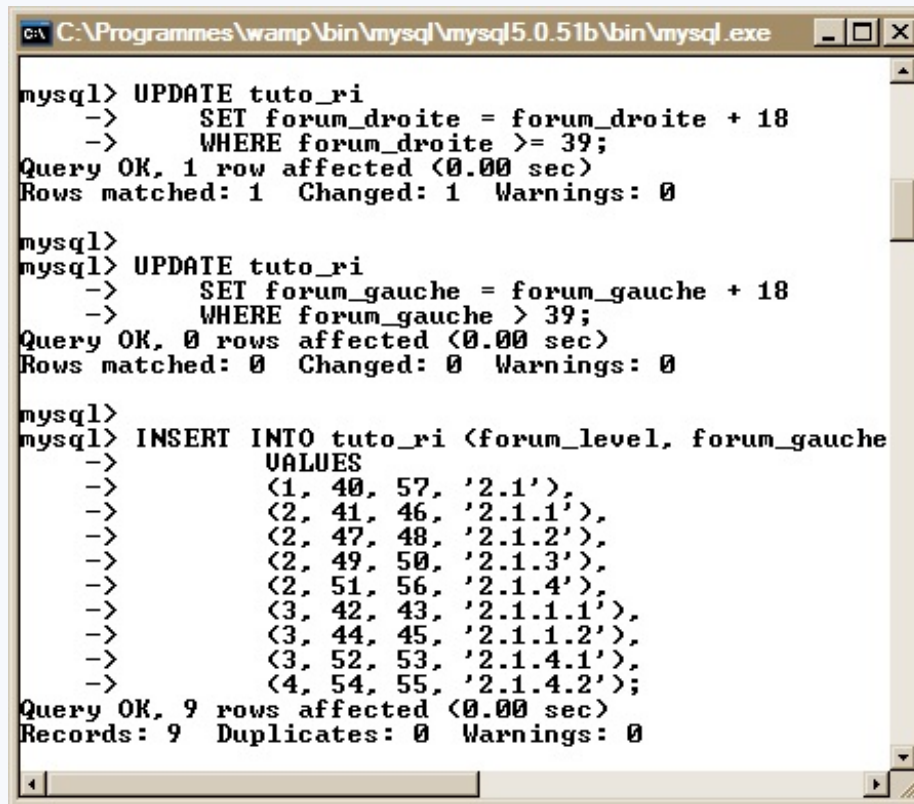
```
UPDATE tuto_ri
  SET forum_droite = forum_droite + 18
  WHERE forum_droite >= 39;

UPDATE tuto_ri
  SET forum_gauche = forum_gauche + 18
  WHERE forum_gauche > 39;

INSERT INTO tuto_ri (forum_level, forum_gauche, forum_droite,
forum_name)
VALUES
  (1, 40, 57, '2.1'),
  (2, 41, 46, '2.1.1'),
  (2, 47, 48, '2.1.2'),
  (2, 49, 50, '2.1.3'),
  (2, 51, 52, '2.1.4'),
  (3, 53, 54, '2.1.1.1'),
  (3, 55, 56, '2.1.1.2'),
  (3, 57, 58, '2.1.4.1'),
  (3, 59, 60, '2.1.4.2');
```



```
(2, 49, 50, '2.1.3'),
(2, 51, 56, '2.1.4'),
(3, 42, 43, '2.1.1.1'),
(3, 44, 45, '2.1.1.2'),
(3, 52, 53, '2.1.4.1'),
(4, 54, 55, '2.1.4.2');
```



```

mysql> UPDATE tuto_ri
->     SET forum_droite = forum_droite + 18
->     WHERE forum_droite >= 39;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> UPDATE tuto_ri
->     SET forum_gauche = forum_gauche + 18
->     WHERE forum_gauche > 39;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql>
mysql> INSERT INTO tuto_ri (forum_level, forum_gauche
->     VALUES
->     (1, 40, 57, '2.1'),
->     (2, 41, 46, '2.1.1'),
->     (2, 47, 48, '2.1.2'),
->     (2, 49, 50, '2.1.3'),
->     (2, 51, 56, '2.1.4'),
->     (3, 42, 43, '2.1.1.1'),
->     (3, 44, 45, '2.1.1.2'),
->     (3, 52, 53, '2.1.4.1'),
->     (4, 54, 55, '2.1.4.2');
Query OK, 9 rows affected (0.00 sec)
Records: 9  Duplicates: 0  Warnings: 0

```

Insertion d'un arbre (noeud) dans un noeud : transformation d'une feuille en un noeud

Bon, c'est vrai, fallait faire des maths : ajouter 39 (borne gauche du nouveau parent, c'est-à-dire la catégorie 2) à toutes les bornes de ce noeud (et à ses éléments fils), pour le rendre "compatible" avec notre arbre récepteur. Ensuite, c'était une insertion des plus banales : il faut décaler les bornes de tous les éléments de notre arbre de.. petit calcul... **18** (à partir de la borne gauche - *non incluse* - du nouveau parent), pour le rendre compatible avec notre arbre. Soit le double du nombre d'éléments à rajouter (9 éléments dans notre cas). Pour les niveaux, il "suffisait" juste de sélectionner les niveaux de l'arbre, et leur additionner le niveau du noeud auquel on veut fixer le nouvel arbre. Or, comme notre noeud "récepteur" a un niveau égal à 0... dans notre cas, ça n'a aucun effet (si ce n'est consommer des ressources). 🤪



Notez qu'ici, puisque la "catégorie 2", en plus d'être simple feuille, est le dernier élément de notre arbre, la seconde requête UPDATE ne sert donc (dans notre cas) à rien, mais je l'ai mise pour vous signaler son existence. Rien de plus, rien de moins 😊.

mysql> SELECT forum_name, forum_gauche, forum_droite
-> FROM tuto_ri
-> ORDER BY forum_gauche ASC;

forum_name	forum_gauche	forum_droite
1	1	38
1.1	2	27
1.1.1	3	24
1.1.1.1	4	23
1.1.1.1.1	5	18
1.1.1.1.1.1	6	13
1.1.1.1.1.1.1	7	8
1.1.1.1.1.1.2	9	10
1.1.1.1.1.1.3	11	12
1.1.1.1.1.2	14	15
1.1.1.1.1.3	16	17
1.1.1.1.2	19	20
1.1.1.1.3	21	22
1.1.2	25	26
1.2	28	29
1.3	30	31
1.4	32	37
1.4.1	33	34
1.4.2	35	36
2	39	58
2.1	40	57
2.1.1	41	46
2.1.1.1	42	43
2.1.1.2	44	45
2.1.2	47	48
2.1.3	49	50
2.1.4	51	56
2.1.4.1	52	53
2.1.4.2	54	55

29 rows in set (0.00 sec)

Et voilà le travail 😊

Voici la suppression (vous pouviez la deviner, quand même 🤪) :

Code : SQL

```
DELETE
  FROM tuto_ri
 WHERE forum_gauche > 39
    AND forum_droite < 58;

UPDATE tuto_ri
  SET forum_gauche = forum_gauche - 18
 WHERE forum_gauche > 39;

UPDATE tuto_ri
  SET forum_droite = forum_droite - 18
 WHERE forum_droite >= 58;
```

C'était tout simplement un mélange entre l'application de la suppression d'une feuille et l'insertion d'un arbre. Comme pour la suppression d'une feuille, les mises à niveau des bornes de l'arbre dans lequel on a supprimé notre noeud ne sont pas obligatoires, mais restent un plus non négligeable pour avoir un arbre propre, et pour pouvoir éviter les trous trop gros dans l'arbre au fur et à mesure des opérations. 🤪

Dans notre cas, comme pour l'insertion, la mise à jour des bornes gauche de tous les éléments n'était pas nécessaire, car notre catégorie 2 se transforme en simple feuille, tout en étant la dernière de notre arbre.

```

mysql> DELETE
-> FROM tuto_ri
-> WHERE forum_gauche > 39
-> AND forum_droite < 58;
Query OK, 9 rows affected (0.00 sec)

mysql>
mysql> UPDATE tuto_ri
-> SET forum_gauche = forum_gauche - 18
-> WHERE forum_gauche > 39;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0 Changed: 0 Warnings: 0

mysql>
mysql> UPDATE tuto_ri
-> SET forum_droite = forum_droite - 18
-> WHERE forum_droite >= 58;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1 Changed: 1 Warnings: 0

```

Suppression d'un sous-arbre (noeud à plusieurs éléments fils) dans un noeud : transformation d'un noeud en une feuille

```

mysql> SELECT forum_name, forum_gauche, forum_droite
-> FROM tuto_ri
-> ORDER BY forum_gauche ASC;

```

forum_name	forum_gauche	forum_droite
1	1	38
1.1	2	27
1.1.1	3	24
1.1.1.1	4	23
1.1.1.1.1	5	18
1.1.1.1.1.1	6	13
1.1.1.1.1.1.1	7	8
1.1.1.1.1.1.2	9	10
1.1.1.1.1.1.3	11	12
1.1.1.1.1.2	14	15
1.1.1.1.1.3	16	17
1.1.1.1.2	19	20
1.1.1.1.3	21	22
1.1.2	25	26
1.2	28	29
1.3	30	31
1.4	32	37
1.4.1	33	34
1.4.2	35	36
2	39	40

```

20 rows in set (0.00 sec)

```

On retourne à notre configuration initiale



Comme pour l'insertion, pour connaître le nombre à soustraire aux deux bornes, il "suffisait" de connaître le nombre d'éléments à supprimer, et à le doubler... Un peu de maths n'a jamais tué personne 🤪 !

C'est "déjà" la fin de ce tutoriel 😊.

Vous savez maintenant utiliser un bel outil nommé la "représentation intervallaire", qui vous permet de vous débrouiller avec aisance pour gérer un super-système de catégories (allant jusqu'à une profondeur très-très-très-(...)très grande (voire infinie)) de ce-que-vous-voulez (l'exemple le plus probant étant celui de [la gestion des tutoriels du Site du Zéro](#)).

Partager



Ce tutoriel a été corrigé par les [zCorrecteurs](#).