T. D. n° 9
Sous-requêtes en MySQL

Résumé

Ce document est le TD n° 9 du module **Base de données**. Il reprend rapidement des éléments du cours et propose une mise en pratique interactive des sous-requêtes.

1 Sous-requêtes

source :`Memecenter`

Reprenez votre base de donnée. Cette base contient *a minima* les tables Animal, Race et Espece.
Nous allons maintenant apprendre à imbriquer plusieurs requêtes, ce qui vous permettra de faire en une seule fois ce qui vous aurait, jusqu'ici, demandé plusieurs étapes.
Une sous-requête est une requête à l'intérieur d'une autre requête. Avec le SQL, vous pouvez construire des requêtes imbriquées sur autant de niveaux que vous voulez. Vous pouvez également mélanger jointures et sous-requêtes. Tant que votre requête est correctement structurée, elle peut être aussi complexe que vous le voulez.
Une sous-requête peut être faite dans une requête de type `SELECT`, `INSERT`, `UPDATE` ou `DELETE`.
La plupart des requêtes de sélection que vous allez voir dans ce TD sont tout à fait réalisables autrement, souvent avec une jointure. Certains préfèrent les sous-requêtes aux jointures parce que c'est légèrement plus clair comme syntaxe, et peut-être plus intuitif. Il faut cependant savoir qu'une jointure sera en général au moins

aussi rapide que la même requête faite avec une sous-requête. Par conséquent, s'il est important pour vous d'optimiser les performances de votre application, utilisez plutôt des jointures lorsque c'est possible.

1.1 Sous-requêtes dans le `FROM`

Lorsque vous faites une requête `SELECT`, le résultat est sous forme de table. Ces tables de résultats peuvent avoir :

- plusieurs colonnes et plusieurs lignes ;
- plusieurs colonnes, mais une seule ligne ;
- plusieurs lignes, mais une seule colonne ;
- ou encore une seule ligne et une seule colonne (c'est-à-dire juste une valeur).

Les sous-requêtes renvoyant plusieurs lignes et plusieurs colonnes ne sont utilisées que dans les clauses `FROM`. Nous allons ici nous intéresser aux trois autres possibilités uniquement.

1.1.1 Comparaisons

L'exemple ci-dessous est une exemple typique d'une sous-requête qui retourne un seul résultat à la requête principale.

```
SELECT *
FROM `table`
WHERE `nom_colonne` = (
    SELECT `valeur`
    FROM `table2`
    LIMIT 1
  )
```

Cet exemple montre une requête interne (celle sur "table2") qui renvoi une seule valeur. La requête externe quant à elle, va chercher les résultat de "table" et filtre les résultats à partir de la valeur retournée par la requête interne.

Une requête imbriquée peut également retourner une colonne entière. Dès lors, la requête externe peut utiliser la commande `IN` pour filtrer les lignes qui possèdent une des valeurs retournées par la requête interne. L'exemple ci-dessous met en évidence un tel cas de figure :

```
SELECT *
FROM `table`
WHERE `nom_colonne` IN (
    SELECT `colonne`
    FROM `table2`
    WHERE `cle_etrangere` = 36
  )
```

1.2 Conditions avec `ANY`, `SOME` et `ALL`

Les conditions avec `IN` et `NOT IN` sont un peu limitées, puisqu'elles ne permettent que des comparaisons de type "est égal" ou "est différent". Avec `ANY` et `ALL`, on va

pouvoir utiliser les autres comparateurs (plus grand, plus petit, etc.).
Bien entendu, comme pour `IN`, il faut des sous-requêtes dont le résultat est soit une valeur, soit une colonne.

- `ANY` : veut dire "au moins une des valeurs".
- `SOME` : est un synonyme de `ANY`.
- `ALL` : signifie "toutes les valeurs".

La requête suivante `ANY` (ou `SOME`) signifie "Sélectionne les lignes de la table A, dont l'id est inférieur à au moins une des valeurs sélectionnées dans la sous-requête".

```
SELECT *

FROM Animal

WHERE espece_id < ANY (

    SELECT id

    FROM Espece

    WHERE nom_courant IN ('Tortue d''Hermann', 'Perroquet amazone')

);
```

Par contre, si vous utilisez `ALL` plutôt que `ANY`, cela signifiera "à toutes les valeurs sélectionnées dans la sous-requête".

À vous !

a) Renvoyer les lignes de la table Animal pour lesquelles l'id de la race vaut 7 via une sous requête.

b) Sélectionner toutes les informations sur les animaux dont l'id est inférieur à tous les id d'espèces dont le nom courant est Tortue d'Hermann, ou Perroquet amazone

c) Sélectionner toutes les informations sur les animaux dont le nom courant de l'espèce est différent d'une Tortue d'Hermann, ou d'un Perroquet amazone (avec une sous requête).

d) Sélectionner les id de la table Espece pour lesquelles le nom_courant est 'Tortue d'Hermann' ou 'Perroquet amazone'). Est il possible d'utiliser la synataxe "=ANY" ?

e) Parmi les femelles perroquets et tortues, on veut connaître la date de naissance de la plus âgée. Formuler votre requête avec une sous-requête (utiliser la fonction `MIN()`.

1.3 Existe et Sous requêtes corrélées

Les conditions `EXISTS` et `NOT EXISTS` s'utilisent de la manière suivante :

```
SELECT * FROM nom_table

WHERE [NOT] EXISTS (sous-requête)
```

Une condition avec `EXISTS` sera vraie (et donc la requête renverra quelque chose) si la sous-requête correspondante renvoie au moins une ligne. Une condition avec `NOT EXISTS` sera vraie si la sous-requête correspondante ne renvoie aucune ligne.

Une sous-requête corrélée est une sous-requête qui fait référence à une colonne (ou une table) qui n'est pas définie dans sa clause `FROM`, mais bien ailleurs dans la requête dont elle fait partie.
Voici un exemple de requête avec une sous-requête corrélée :

```
SELECT colonne1

FROM tableA

WHERE colonne2 IN (

    SELECT colonne3

    FROM tableB

    WHERE tableB.colonne4 = tableA.colonne5

    );
```

Ici, seule la tableB est sélectionnée dans la clause `FROM`, il n'y a pas de jointure avec la tableA, et pourtant, on utilise la tableA dans la condition.
Puisque la clause `FROM` de la requête principale sélectionne la tableA. La sous-requête est donc corrélée à la requête principale.

À vous !

a) Affichez toutes les races s'il existe un animal qui s'appelle Balou.

b) Sélectionner toutes les races dont on ne possède aucun animal.

c) Sélectionner tous les animaux ayant au moins un enfant

2 Jointures et sous-requêtes : modification de données

Vous allez apprendre ici à utiliser ces outils, non pas dans le cadre de la sélection de données, comme on l'a fait jusqu'à présent, mais pour :

- l'insertion : les sous-requêtes vont permettre d'insérer des données dans une table à partir de données venant d'autres tables (mais pas exclusivement) ;
- la modification : jointures et sous-requêtes vont permettre non seulement d'utiliser des critères de sélection complexes, mais également d'aller chercher les nouvelles valeurs dans d'autres tables ;

— la suppression de données : comme pour la modification, les critères de sélection pourront être plus complexes grâce aux jointures et sous-requêtes.

2.1 INSERT INTO... SELECT

Cette syntaxe permet de sélectionner des éléments dans des tables, afin de les insérer directement dans une autre.

```
INSERT INTO nom_table

   [(colonne1, colonne2, ...)]

SELECT [colonne1, colonne2, ...]

FROM nom_table2

[WHERE ...]
```

Vous n'êtes bien sûr pas obligé de préciser dans quelles colonnes se fait l'insertion, si vous sélectionnez une valeur pour toutes les colonnes de la table. Ce sera cependant rarement le cas puisque nous avons des clés primaires auto-incrémentées.
Avec cette requête, il est absolument indispensable (sinon l'insertion ne se fera pas) d'avoir le même nombre de colonnes dans l'insertion et dans la sélection, et qu'elles soient dans le même ordre.
Si vous n'avez pas le même nombre de colonnes, cela déclenchera une erreur. De plus, si l'ordre n'est pas bon, vous aurez probablement une insertion erronée.
Voici un exemple

```
INSERT INTO Animal

    (nom, sexe, date_naissance, race_id, espece_id)

SELECT  'Nom', 'Sexe', 'date_naissance', id AS race_id, espece_id

    -- Attention à l'ordre !

FROM Race WHERE nom = 'nom de la race';
```

À vous !

a) Ajoutez Yoda, un chat mâle maine coon née le 2010-11-09. Vous ne vous connaissez pas de l'id de la race maine coon ni de celui de l'espèce Chat. Vous devez utiliser une sous-requête pour insérer directement l'id de la race et de l'espèce à partir du nom de la race.

b) Sélectionnez maintenant les maine coon de notre base, pour vérifier que l'insertion s'est faite correctement (utiliser un `INER JOIN`)

2.2 Élément à modifier

Il est tout à fait possible de modifier une table dont vous ingorez des informations partielles grâce à des sous-requête :

```
UPDATE TableA SET col_A1 =

    (SELECT col_B1 FROM TableB WHERE col_B2 = 'valeur B2')

WHERE col_A2 = 'valeur A2';
```

Il est bien entendu indispensable que le résultat de la sous-requête soit une valeur ! Une limitation importante des sous-requêtes est que l'on ne peut pas modifier un élément d'une table que l'on utilise dans une sous-requête.

Vous pouvez également utiliser `UPDATE` avec des jointures.
Voici la syntaxe que vous devriez utiliser :

```
UPDATE Table_A       -- Classique !

INNER JOIN Table_B   -- Jointure.

    ON Table_A.col_1= Table_B.col_1

    -- Condition de la jointure.

SET Table_A.col_2 = Table_B.col_2

    -- Ensuite, la modification voulue.

WHERE Table_A.col_2 IS NULL

    -- Exemple de condition

AND Table_A.col_3 IN ('bla, 'blabla');

    -- Conditions suplémentaires (facultatif)
```

À vous !

a) Faite en sorte que tous les perroquets aient en commentaire : "Coco veut un gâteau !" sachant que vous ne savez pas que l'id de cette espèce est 4.

b) Ajoutez la race 'Nebelung', d'especе_id 2 (chat), dont le commentaire est 'Chat bleu russe, mais avec des poils longs...'

c) Vous découvrez que Cawette n'est pas un bleu russe, mais un nebelung. Vous devez donc changer sa race. Vous avez besoin de l'id de la race nebelung que vous venez d'ajouter. Réalisez cette modification grâce à une sous-requête.

d) Changez également la race de Callune pour qu'il soit un Nebelung via une sous requête

e) Vérifiez que Cawette et Callune sont bien des nebelung

f) Pour les tortues et les perroquets, si un animal n'a pas de commentaire, ajouter comme commentaire la description de l'espèce, en utilisant une jointure.

2.3 Suppression

L'utilisation des sous-requêtes et des jointures est assez ressemblante concernant la suppression et la modification. Simplement, pour la suppression, les sous-requêtes et jointures ne peuvent servir qu'à sélectionner les lignes à supprimer.
On peut, tout simplement, utiliser une sous-requête dans la clause `WHERE`.

```
DELETE FROM Table_A

WHERE col_1= "valeur A"   AND col_2=

    (SELECT col_2 FROM Table_B WHERE col_1= "valeur B");
```

Pour les jointures, c'est le même principe. Si je reprends le même problème que ci-dessus, voici comment supprimer la ligne voulue avec une jointure :

```
DELETE Table_A   -- Je précise de quelles tables les données
   doivent être supprimées

FROM Table_A     -- Table principale

INNER JOIN Table_B ON Table_A.id = Table_B.id

    -- Jointure

WHERE Table_A.col1 = 'valeur A'

    AND Table_B.col1 = 'valeur B';
```

À vous !

a) Supprimez le chat Carabistouille via une sous-requête. Attention, il y a également un perroquet du même nom dans la table Animal.

b) Rajoutez le chat Carabistouille via une sous-requête.

c) Supprimez le chat Carabistouille via une jointure