BASE DE DONNEES ET SYSTEMES DE GESTION DE BASE DE DONNEES (SGBD)

CHAPITRE VII : NOTION DE SOUS REQUETE

Lorsque les requêtes deviennent plus complexes, il peut s'avérer nécessaire d'imbriquer les interrogations. Un SELECT imbriqué sera en fait un bloc SELECT… FROM… WHERE appelé bloc de qualification, introduit dans la clause WHERE d'un bloc.

Ce type d'imbrication sera utilisé lorsque, dans la condition d'une clause WHERE, on désire indiquer qu'une colonne prend ses valeurs dans une liste. Cette liste sera obtenue par le bloc de qualification et sera ensuite utilisée par le bloc externe. La liaison entre blocs sera réalisée par des opérateurs, à savoir ALL, IN, ANY, EXISTS, et par les opérateurs de comparaison classiques (=, >, <, …).

SOUS-REQUETE SIMPLE

L'exemple suivant permet de rechercher les noms des personnes qui habitent la même localité que M. Durant :

```
SELECT nom, prenom, localite
        FROM passagers WHERE localite IN
(SELECT localite FROM
        passagers WHERE nom = 'Durant') ;
```

Précisons l'ordre d'évaluation de la requête :

1- La sous-requête interne est évaluée : la localité associée au nom Durant est recherchée dans la table PASSAGERS ;
2- La requête externe est évaluée : les noms des personnes habitant la même localité que Durant sont trouvés.

SOUS-REQUETE FAISANT INTERVENIR DEUX TABLES

L'exemple ci-dessous détermine les numéros de téléphone des personnes qui prennent le vol SN737, pour les prévenir en cas de retard éventuel du vol :

```
SELECT tel FROM passagers
    WHERE no_pass IN
        (SELECT no_pass FROM reservations
            WHERE no_vol = 'SN737') ;
```

Sans une sous requête, on a :

```
SELECT tel FROM passagers p, reservations r
    WHERE r.no_pass = p.no_pass
        AND r.no_vol = 'SN737' ;
```

Ici, la table du bloc interne est RESERVATIONS alors que celle du bloc externe est PASSAGERS.

La requête interne est d'abord évaluée, et il en ressort un ensemble de numéros de personnes, à savoir celles qui prennent le vol SN737. A ce moment, la requête équivaut à :

```
SELECT tel FROM passagers
    WHERE no_pass IN (n1, n2, …, np) ;
```

qui constitue un SELECT CLASSIQUE.

Dans cet exemple, les noms des colonnes n'ont pas été précédés du nom de la table associée, alors que no_pass apparaît dans deux tables distinctes.

La règle est la suivante : la table implicitement associée à un nom de colonne est celle qui apparaît dans la clause FROM du SELECT de même niveau. En d'autres termes, l'exemple donné équivaut à :

```
SELECT passagers.tel
        FROM passagers
            WHERE passagers.no_pass IN
                (SELECT reservations.no_pass
                    FROM reservations
                        WHERE reservations.no_vol = 'SN737');
```

Ou

```
SELECT p.tel
    FROM passagers p
        WHERE p.no_pass IN
            (SELECT r.no_pass
                FROM reservations r
                    WHERE r.no_vol = 'SN737');
```

Note

Dans l'utilisation des sous-requêtes, les synonymes sont d'usage fréquent et ils facilitent grandement l'écriture.

SOUS-REQUETE A IMBRICATIONS MULTIPLES

Un nombre supérieur d'imbrications est présenté à l'exemple suivant. Celui-ci fournit cette fois les noms des passagers inscrits sur le même vol que M. Dupuis.

```
SELECT nom, prenom FROM passagers
```

```
WHERE no_pass IN
    (SELECT no_pass FROM reservations
        WHERE no_vol IN
            (SELECT no_vol FROM reservations
             WHERE no_pass IN
                 (SELECT no_pass FROM passagers
                  WHERE nom = 'Dupuis'))) ;
```

Etapes :

- Recherche du passager de nom Dupuis et détermination de son no_pass (bloc le plus interne) ;
- Recherche du no_vol correspondant au no_pass trouvé ;
- Recherche du no_pass des réservations correspondant au no_vol trouvé ;
- Recherche du nom des passagers dont le no_pass est parmi ceux qui ont été trouvés.

SOUS-REQUETE CORRELATIVE

Jusqu'à présent, il était possible d'évaluer les valeurs du bloc de qualification interne avant d'évaluer le bloc externe. Ce n'est cependant pas toujours réalisable, comme le montre l'exemple suivant :

Recherchons le numéro de téléphone des personnes qui prennent le vol SN737 ;

```
SELECT tel FROM passagers
    WHERE 'SN737' IN
        (SELECT no_vol FROM reservations
            WHERE no_pass = passagers. no_pass) ;
```

A la dernière ligne, la première référence à no_pass se rapporte implicitement à la table RESERVATIONS alors que la deuxième référence se rapporte explicitement à PASSAGERS. Dans cet exemple, il n'est pas possible d'évaluer en premier lieu la requête interne. L'évaluation suit ici la procédure suivante :

1. Le premier tuple de la table externe (PASSAGERS) est examiné et la colonne passagers.no_pass a alors une valeur (par exemple 18). Il reste alors à évaluer le bloc interne qui prend la forme :

   SELECT no_vol FROM reservations

   WHERE no_pass = 18 ;

On obtient un ou plusieurs résultats qui permettent d'achever l'évaluation de la requête externe : si l'un des résultats vaut SN737, le numéro 18 fera partie de la réponse.

2. Ce traitement est répété pour les autres tuples de la table PASSAGERS.

Remarque

Dans les exemples ci-dessus, l'opérateur qui fait la liaison entre les blocs est l'opérateur IN. Il est employé lorsque le bloc interne fournit plusieurs valeurs au bloc externe.

Dans certains cas, on est sûr de n'obtenir qu'une seule valeur en réponse. Dans ce cas, les opérateurs de comparaison traditionnels (>, >=, <, ...) peuvent être utilisés.

SOUS-REQUETE UTILISANT L'OPERATEUR ALL

Dans l'exemple ci-dessous, l'opérateur ALL permet de sélectionner les tuples de la table AVIONS pour lesquels la valeur de nbre_passag est supérieure à celle des Boeing 727.

```
SELECT * FROM avions WHERE nbre_passag > ALL
    (SELECT nbre_passag FROM avions WHERE modele LIKE '727%') ;
```

Pour MySql utiliser :

```
SELECT * FROM avions WHERE nbre_passag >
        (SELECT nbre_passag FROM avions WHERE modele LIKE '727%') ;
```

La syntaxe générale de sous-requêtes qui utilisent l'opérateur ALL est :

```
… nom_col op ALL (SELECT… FROM…) ;
```

"op" est un opérateur de comparaison classique (=, ¬, = (! = ou ^=), >, <, >=, <=).

L'interprétation de la sous-requête se fait comme suit :

- Le SELECT interne est évalué et un ensemble de valeurs (n1, n2, …, np) est mémorisé dans une table temporaire.
- Les tuples qui vérifient la condition "nom_col op ni" pour l'ensemble des ni, i = 1,…, p, sont sélectionnés.

Note

La condition ¬ = ALL équivaut à NOT IN.

SOUS-REQUETE UTILISANT L'OPERATEUR ANY

Une requête employant le mot clé ANY sera évaluée de la même manière que pour le mot-clé ALL, si ce n'est que la condition "nom_col op ni" doit être réalisée par une quelconque des valeurs n1, …, np issues du SELECT interne.

Recherchons les numéros d'identification des avions dont l'envergure est supérieure à un quelconque "Boeing".

```
SELECT type FROM avions
    WHERE enverg > ANY
        (SELECT enverg FROM avions
            WHERE marque = 'Boeing');
```

Note : La condition = ANY est équivalente au mot-clé IN.

SOUS-REQUETE UTILISANT L'OPERATEUR EXISTS

La forme générale des sous-requêtes utilisant EXISTS est :

```
…WHERE EXISTS (SELECT … FROM T WHERE …) ;
```

Si au moins une ligne de la table T satisfait à la condition de la sous-requête, celle-ci donnera un résultat non vide, auquel cas la clause WHERE sera considérée comme satisfaite.

Par exemple, recherchons les modèles des avions repris pour un vol :

```
SELECT modele FROM avions
    WHERE EXISTS
        (SELECT * FROM vols
            WHERE no_av = avions.type) ;
```

EXISTS, combiné avec un opérateur logique tel que NOT, peut servir à exprimer des questions telles que celle-ci : « Quels sont les avions pour lesquels aucune réservation n'est prévue ? »

```
    SELECT no_av FROM vols
        WHERE NOT EXISTS
            (SELECT * FROM reservations
                WHERE   no_vol= vols.no_vol) ;
```

Ou

```
    SELECT no_av FROM vols
        WHERE  ! EXISTS
```

```
(SELECT * FROM reservations
        WHERE   no_vol= vols.no_vol) ;
```

Ce type de question nous amène ainsi à considérer un autre opérateur de logique mathématique : le quantificateur universel FORALL. Ce dernier apparaît dans des prédicats de la forme :

FORALL x (P(x))

Cette expression sera vraie si et seulement si P(x) est vrai pour toute valeur possible de x. Par exemple, si X est compris entre 10000 et 20000, FORALL x (x > 0) est vrai.

Alors que :

FORALL x (x>15000) est faux.

Comme l'opérateur FORALL n'existe pas en SQL, on le remplace par une combinaison des opérateurs NOT et EXISTS. En effet, si P(x) est vrai pour tout x, cela revient à dire qu'il n'existe pas de valeur de x pour laquelle P(x) est faux.

En termes de logique, on peut donc écrire :

FORALL x (P(x)) = NOT (EXISTS x (NOT (P(x)).

Illustrons le concept de FORALL par exemple. Recherchons les passagers qui ont une réservation sur chaque vol :

```
SELECT nom FROM passagers
    WHERE NOT EXISTS
        (SELECT * FROM vols
         WHERE NOT EXISTS
```

```
                (SELECT * FROM  reservations
                        WHERE no_vol =  vols.no_vol
                                AND no_pass  = passagers.no_pass)) ;
```

Ou

```
SELECT nom FROM passagers
        WHERE !EXISTS
                (SELECT * FROM vols
                 WHERE !EXISTS
                        (SELECT * FROM  reservations
                                WHERE no_vol =  vols.no_vol
                                        AND no_pass  = passagers.no_pass)) ;
```

Explication :

Dans la requête la plus externe, on passe en revue tous les tuples de la table PASSAGERS. Prenons un passager quelconque, celui dont le numéro est 217, par exemple. Pour chaque passager, on passe en revue tous les VOLS. Soit SN777 le numéro de ces vols.

On arrive alors dans la sous-requête la plus interne. On y cherche les tuples de la table RESERVATIONS pour lesquels no_pass vaut 217 et no_vol vaut SN777. Si l'on en trouve, la clause WHERE NOT EXISTS de la sous-requête liée à la table VOLS ne sera pas satisfaite et le tuple en question ne fera pas partie de la réponse. On passe ainsi en revue tous les vols pour le passager 217. Si ce dernier a une réservation sur chaque vol, aucun tuple ne satisfera le SELECT intermédiaire, ce qui donnera une valeur VRAI à la clause WHERE NOT EXISTS du SELECT externe qui affichera le nom de ce passager comme réponse finale.

Si, par contre, le passager 217 n'a pas de réservation prévue sur au moins un vol, le bloc le plus interne ne donnera pas de réponse pour ce vol, ce qui satisfait la condition

NOT EXISTS de la clause WHERE intermédiaire et donc rend FAUX le NOT EXISTS externe. Le nom du passager ne sera donc pas affiché.

En raisonnant de cette manière pour chaque passager, on obtiendra bien en réponse les noms des passagers souhaités.

SOUS-REQUETE ET CREATION DE TABLE

Des versions récentes de SQL offrent la possibilité de créer de tables au départ de tables déjà existant, par exemple pour des utilisations très spécifiques. Dans ce cas, il est permis de définir la nouvelle table et d'y transférer des données voulues à l'aide d'une seule commande dont la syntaxe est :

```
CREATE TABLE nom_nouv_table
        (nom_coll  [, nom_col2,..])
                AS  (SELECT nom_coll[, nomcol2,…]
                        FROM nom_anc_table
                                WHERE condition) ;
```

Notons qu'il n'est pas nécessaire de redéfinir les types des colonnes de la nouvelle table, types déjà présents dans la table de référence.

Par exemple, si une agence s'occupe exclusivement des passagers habitant Belgique, elle créera une table appelée PASS_BELG au départ de la table PASSAGERS à l'aide de la commande :

```
CREATE TABLE pass_belg
        (SELECT no_pass, nom, prenom, rue, boite, code,
                localite, tel, nopasspt FROM passagers
                        WHERE pays ='B') ;
```

- Citons pour mémoire la possibilité d'exprimer l'implication logique en SQL :

  En logique, l'implication s'écrit : si p alors q, ce qui se définit par : Si p alors q = NOT (p) OR q,

  p et q étant des prédicats.

- En général, il est possible de répondre à une même question de plusieurs manières. Les solutions sont équivalentes et dépendront uniquement de l'utilisateur, suivant qu'il cherche la rapidité de recherche ou la facilité de compréhension des requêtes.

Les éléments essentiels de la clause SELECT ont ainsi été passés en revue. Des exemples concrets liés à ces définitions se trouvent exposés dans les chapitres suivants et sont en mesure de clarifier les idées.