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

CHAPITRE IV : RECHERCHES SIMPLES

Sommaire

INTRODUCTION

Les instructions d'interrogation de données sont les ordres les plus utilisés en SQL. Elles permettent :

- de sélectionner certaines colonnes d'une ou plusieurs tables en fonction de certains critères ;
- d'extraire certains tuples (enregistrements) et de les trier en fonction de la valeur de certains attributs ;
- d'utiliser des fonctions arithmétiques et de groupement pour des calculs.

La commande SELECT réalise ces fonctions. Dans sa forme la plus générale, et sans en détailler les éléments, elle peut s'écrire :

```
SELECT    [DISTINCT| ALL]    select-list

                    [,select- list,...]

FROM <nom_table1 | nom_vuel >  nom_corrélation1,...]

[,< nom_table2 |nom_vue2> nom_corrélation2 ,..]

WHERE     condition

GROUP  BY <nom_col1 | entier>
          [ ,   < nom_col2 | entier ,...]
HAVING  condition

FOR UPDATE OF  nom_col1   [ , nom_col2 ,...]

ORDER  BY <nom_col1 | entier> [ASC | DESC]

          [ , < nom_col2 | entier> ASC  | DESC ] ,...];
```

Les clauses SELECT et FROM sont, seules obligatoires.

Nous allons analyser progressivement cette commande depuis la recherche simple sans condition jusqu'aux recherches imbriquées avec tri et groupement.

RECHERCHE SIMPLE SANS CONDITION

Le diagramme syntaxique se ramène à :

```
SELECT   nom_col1  [ , nom_col2 ,...]
      FROM   nom_table  ;
```

Cette forme, la plus simple du SELECT, permet d'extraire de la table "nom_table" les colonnes "nom_col1", "nom_col2",... Cette liste de colonnes est en effet la "select-list" typique.

Exemple :

```
SELECT   no_pass, nom, prenom
        FROM  passagers ;
```

Il est possible de visualiser toutes les colonnes d'une table en remplaçant la liste de ces colonnes par "*". Ce sigle constitue donc la deuxième forme de "select-list".

Ainsi,

```
SELECT   *
        FROM   avions ;
```

Fournit le contenu de la table avions dans son entièreté, les attributs, tant dans le même ordre que lors de la création de la table.

LA CLAUSE DISTINCT

Si l'on souhaite ne pas voir apparaître de lignes redondantes dans la table résultante, la clause DISTINCT (UNIQUE pour certaines versions de SQL) sera ajoutée. Si cette clause n'apparaît pas, l'option ALL est prise par défaut et tous les tuples sont affichés.

Exemple :

```
SELECT    nom

        FROM    passagers ;
```

Donnera par exemple la table :

```
Nom
-----------------------------------
Dulieu
Lagaffe
Lagaffe
Michelin
```

Par contre, l'instruction :

```
SELECT  DISTINCT  nom

        FROM  passagers ;
```

Fournira le résultat :

```
Nom
------------------
Dulieu
Lagaffe
Michelin
```

Remarque :

Le SELECT DICTINCT ne peut pas être associé à une colonne de caractères de type LONG VARCHAR.

Il est permis de citer plusieurs tables dans la clause FROM. Cela oblige alors à identifier les noms de colonnes à l'une ou l'autre table de manière non ambiguë.

Si un même nom de colonne apparaît dans deux tables distinctes, il sera nécessaire de le faire précéder du nom de table associée, dans la clause SELECT, pour que le nom ainsi formé soit unique.

Ainsi, la colonne PAYS est présente dans les tables passagers et avions. L'instruction :

```
SELECT  passagers.pays, modele
        FROM  passagers, avions ;
```

Fournit la colonne pays de la table passagers et la colonne modele de la table avions, sans ambiguïté possible.

LE NOM DE CORRELATION OU SYNONYME

Notons que dans la clause FROM, un "nom de corrélation" ou alias peut être indiqué à la suite d'un nom de table ou de vue. Il permet de raccourcir les écritures lorsqu'un nom de table survient à plusieurs reprises dans une instruction.

Exemple :

```
SELECT  passagers.no_pass, passagers.nom,

      reservations.no_pass, avions.modele

         FROM  passagers, reservations, avions ;
```

Il est plus agréable d'écrire :

```
SELECT  P.no_pass, P.nom, R.no_pass, A.modele

         FROM  passagers P, reservations R, avions A ;
```

Ou

```
SELECT  P.no_pass, P.nom, R.no_pass, A.modele

         FROM  passagers AS P, reservations AS R, avions AS A ;
```

L'intérêt des noms de corrélation se révèlera évident lors de l'utilisation de sous-requêtes de conditions de recherche.

RECHERCHE SIMPLE AVEC CONDITION

SELECT nom_coll [,nom_col2 ,...]

FROM nom_table1 [,nom_table2 ,...]

WHERE condition ;

La clause WHERE permet de sélectionner dans la table obtenue par le SELECT ...FROM ... les tuples répondant à un critère précis.

La "condition" de la clause WHERE est une expression logique qui peut contenir :

- les colonnes des tables citées dans le FROM ;
- les opérateurs de comparaison <,<=,=,>,>=,!=,!<;("!" peut être remplacé par NOT dans ce type d'opérateurs ou par "^" dans certains langages);
- les opérateurs logiques NOT, AND et OR ;
- les opérateurs ensemblistes BETWEEN, IS NULL, LIKE, EXISTS et IN.

Remarque

L'introduction de parenthèses permet d'imposer l'ordre d'évaluation de l'expression, qui prend parfois une forme complexe.

Exemples de requêtes simples :

1- SELECT marque, modele FROM avions
WHERE pays = 'USA';

2- SELECT marque, modele FROM avions
WHERE auto >=10000 ;

3- SELECT marque, modele FROM avions
WHERE PAYS = 'USA'
AND (auto >= 10000 OR leng > 30) ;

OPERATEURS ENSEMBLISTES

• L'opérateur BETWEEN permet de comparer une valeur avec un intervalle. Sa forme est la suivante :

... expression [NOT] BETWEEN expression AND expression ...

Ainsi, on peut sélectionner les vols prévus à une époque particulière :

SELECT no_vol, date_dep FROM vols
WHERE date_dep BETWEEN '1988-09-01' AND '1988-09-30';

Pour MySQL :

```
SELECT no_vol, date_dep FROM vols
    WHERE date_dep BETWEEN
        CAST('1988-09-01' AS DATE) AND CAST('1988-09-30' AS DATE);
```

OU

```
SELECT no_vol, date_dep FROM vols
    WHERE (date_dep >= CAST('1988-09-01' AS DATE)
        AND date_dep <= CAST('1988-09-30' AS DATE));
```

• L'opérateur IN teste l'appartenance d'une valeur à un ensemble de valeurs. Cet ensemble peut être une liste explicite de valeurs ou le résultat d'une sous-requête.

Sa syntaxe est de la forme :

... expression [NOT] IN < EXPRESSION | (sous-requête) > ...

La liste des vols partant des USA s'obtiendra par :

```
SELECT no_vol, origine FROM vols

    WHERE origine IN ('BOS','ATL','NYC') ;
```

Ou

```
SELECT no_vol, origine FROM vols

    WHERE origine = 'BOS' OR origine = 'ATL' OR origine = 'NYC' ;
```

Note :

La table des aéroports de l'annexe 5 donne : BOS=Bostone, ATL=Atlanta, NYC=New York City.

• L'opérateur LIKE compare des chaînes de caractères avec un modèle.

Les caractères spéciaux "%" et "_" qui remplacent respectivement toute chaîne et tout caractère peuvent être insérés dans la chaîne modèle.

La syntaxe de cet opérateur est :
... nom _colonne [NOT] LIKE 'modele de chaîne' ...

Trouvons les numéros des passagers dont le nom commence par "D" :

```
SELECT no_pass, nom  FROM passagers

    WHERE nom LIKE 'D%' ;
```

Les noms des passagers dont le prénom contient les lettres ARP à partir de la troisième position :

```
SELECT nom, prenom FROM passagers

    WHERE nom LIKE '__arp%' ;
```

Les noms des passagers dont le prénom contient la lettre « r » en deuxième position :

```
SELECT nom, prenom FROM passagers WHERE prenom LIKE '_r%' ;
```

Les noms des passagers dont le prénom contient la lettre « r » :

```
SELECT nom, prenom FROM passagers WHERE prenom LIKE '%r%' ;
```

• L'opérateur IS NULL teste l'existence de valeurs nulles pour un attribut. On écrit :

```
...nom_colonne  IS  [NOT]  NULL ...
```

Par exemple, tous les vols sans escale s'obtiennent par la requête :

```
SELECT  no_vol, escale  FROM  vols
        WHERE  escale  IS NULL ;
```

Note :

Toute comparaison entre une chaîne et NULL donne un résultat FAUX. Ainsi, nom != 'Liege' donne FAUX, même si nom vaut NULL.

• L'opérateur EXISTS teste l'exercice de certains tuples dans une table. Sa structure est la suivante :

```
...EXISTS   (sous-requête) ...
```

Sa description et les exemples associés seront développés au chapitre 7 lorsque la notion de sous-requête aura été abordée.

RECHERCHE FAISANT INTERVENIR PLUSIEURS TABLES

La recherche d'informations provenant de plusieurs tables est une des opérations les plus fondamentales dans le modèle relationnel ; elle repose sur la notion de JOINTURE. Plusieurs types de jointures existent en SQL et ces variantes contribuent grandement à sa puissance.

• On parlera de jointure lorsque le résultat de la requête provient de plusieurs tables. La jointure qui fait intervenir deux tables est fréquente et, en particulier, l'EQUI-JOINTURE se caractérise par l'apparition, dans la clause WHERE, d'une relation d'égalité entre les deux tables.

Prenons par exemple les tables PASSAGERS ET RESERVATIONS. On souhaite par exemple connaître la destination de différents passagers. La requête s'écrit :

```
SELECT  nom, destin
        FROM  passagers, reservations
                WHERE  passagers.no_pass = reservations.no_pass ;
```

La jointure des deux tables est réalisée par l'argument de la clause WHERE qui impose l'égalité de

colonnes appartenant chacune à une table.

Note

La spécification d'une condition de jointure est nécessaire pour éviter un Produit Cartésien, c'est-à-dire l'opération qui fournit tous les tuples possibles résultant de la jointure de plus d'une table. Ainsi, par exemple, on aurait pu écrire la requête précédente sous la forme :

```
SELECT  passagers.no_pass, destin

FROM  passagers, vols ;
```

En supposant que la relation PASSAGERS contienne 400 tuples et que la relation VOLS en contienne 100, comme le produit cartésien donnerait toutes les combinaisons, 40 000 tuples seraient produits, ce qui n'est pas souhaitable.

D'autres requêtes peuvent faire intervenir trois tables. Par exemple, on peut trouver les modèles d'avions empruntés par les différents passagers.

Les noms des passagers associés aux modèles d'avions s'obtiennent par la commande :

```
SELECT  nom, modele

        FROM  passagers, reservations, vols, avions

                WHERE  passagers.no_pass = reservations.no_pass

                        AND  vols.no_av = avions.type ;
```

Pour trois tables, deux jointures interviennent. En généralisant, on constate que n-1 jointure suffisent pour n tables.

• La NON EQUI-JOINTURE lie des tables entre elles par des opérateurs autres que l'égalité. Par exemple, les opérateurs >, <, != (^= ou <>), BETWEEN, LIKE peuvent être utilisés.

Exemple :

```
SELECT nom, vols.no_vol , vols.date_dep
        FROM passagers, reservations, vols
                WHERE passagers.no_pass = reservations.no_pass
                        AND vols.date_dep > CAST('1988-04-01' AS DATE) ;
```

• L'AUTO-JOINTURE est la jointure d'une table par elle-même. Dans ce cas, pour faire la distinction entre les différentes occurrences d'une même table, on définit des SYNONYMES ou NOMS DE CORRELATION déjà vus précédemment.

Par exemple, on peut souhaiter connaître les personnes qui habitent la même ville que Simon :

```
SELECT P.nom, P.localite FROM passagers P, passagers Q
        WHERE P.localite = Q.localite AND Q. nom = 'Simon' ;
```

Si l'on désire lister tous les passagers pour chaque vol donné, il se pourrait qu'un ou plusieurs vols

n'aient pas encore fait l'objet d'une réservation, auquel cas la jointure est impossible. Pour résoudre ce problème, on utilise un dernier type de jointure appelé la JOINTURE EXHAUSTIVE. Son principe est de forcer l'apparition, dans la table résultante, des tuples d'une table auxquels aucun tuple d'une autre table ne correspond. Pour cela, le signe (+) est ajouté au nom de la colonne réalisant la jointure et pour laquelle la condition de jointure pourrait ne pas être satisfaite.

On écrira, pour afficher les noms des passagers associés aux divers vols :

```
SELECT   nom, no_vol
    FROM   passagers, reservations
        WHERE   passagers.no_pass = reservations.no_pass(+) ;
```

Le (+) après reservations.no_pass engendre l'ajout d'une ligne supplémentaire contenant des valeurs nulles dans la table reservations. Cette ligne est jointe aux tuples de la table passagers qui n'ont pas de réservation associée. La totalité des passagers est ainsi affichée.

Pour afficher tous les passagers non repris sur un vol, l'instruction suivante joint les tables et recherche ensuite les égalités avec lignes NULL :

```
SELECT   nom, no_vol
    FROM   passagers, reservations
        WHERE   passagers.no_pass = reservations.no_pass (+)
            AND   reservations.no_vol   IS  NULL ;
```

REMARQUE
Généralement, les jointures exhaustives unidirectionnelles sont, seules, acceptées ; on peut avoir (+) sur les deux noms réalisant la jointure.

Note

Il est clair que les jointures présentées ici comme s'appliquant à des tables, peuvent aussi bien faire intervenir des vues, susceptibles de se combiner entre elles ou avec une ou plusieurs tables.

CONCLUSION

Les recherches simples en SQL sont donc réalisables par des questions de forme SELECT… FROM… WHERE qui nécessitent en outre des opérations ensemblistes ou des opérateurs de comparaison, et font très souvent intervenir l'une ou l'autre forme de jointure.

Les extensions de ces composantes de base font l'objet des chapitres suivants.