CHAPITRE 6

Langages de manipulation relationnels: SQL

SQL (Structured Query Language) est le langage de manipulation des données relationnelles le plus utilisé aujourd'hui. Il est devenu un standard de fait pour les relationnels. Il possède des caractéristiques proches de l'algèbre relationnelle (jointure par emboîtement) et d'autres proches du calcul des tuples (variables sur les relations).

Les exemples dans ce chapitre s'appuient sur la base de données relative aux fournisseurs (F), produits (P), usines (U) et livraisons (PUF), décrite par le schéma suivant:

F (NF, nomF, statut, ville)
P (NP, nomP, poids, couleur)
U (NU, nomU, ville)
PUF (NP, NU, NF, qt)

1. Format de base d'une requête

SELECT Liste des noms d'attributs du résultat
FROM Nom d'une relation (ou de plusieurs relations)
[WHERE Condition logique qui définit les tuples du résultat]

Exemple: nom et poids des produits rouges.

```
SELECT  nomP, poids
FROM P
WHERE   couleur = "rouge"
```

Exemple : tous les renseignements sur tous les fournisseurs.

```
SELECT  NF, nomF, statut, ville
FROM F
ou
SELECT  *  /* l'étoile signifie: tous les attributs*/
FROM F
```

Un résultat sans doubles

Les SGBD commercialisés (dont les SQL...) ne suppriment pas automatiquement les doubles. La clause DISTINCT permet à l'utilisateur d'avoir un résultat sans double.

Exemple : liste des couleurs qui existent.

```
SELECT DISTINCT  couleur
FROM   P
```

Un résultat trié

La clause ORDER BY permet de définir un ordre de tri pour les tuples du résultat.

Exemple : liste des fournisseurs de Lausanne par ordre alphabétique.

```
SELECT  nomF, NF, statut
FROM F
WHERE   ville = "Lausanne"
ORDER BY  nomF  ASC , NF  ASC
```

Si plusieurs fournisseurs ont le même nom, ils seront classés selon leurs numéros.

2. Recherche avec blocs emboîtés

Exemple : numéros des fournisseurs de produits rouges ?

ensemble des numéros des produits rouges :

```
SELECT  NP
FROM      P
WHERE couleur = "rouge"
```

ensemble des numéros des fournisseurs de produits rouges :

```
SELECT NF
FROM   PUF
WHERE NP IN
        ( SELECT  NP
          FROM P
          WHERE couleur = "rouge" )
```

Le mot clef IN signifie "appartient", l'opérateur mathématique de la théorie des ensembles ($\in$).
La phrase NP IN (SELECT NP FROM P WHERE couleur = "rouge") est une condition logique, signifiant "la valeur de NP est dans l'ensemble des numéros de produits rouges", ce qui est vrai ou faux.

Pour répondre à cette requête, le SGBD :
1. exécute la requête interne (calcul de l'ensemble des numéros des produits rouges),
2. exécute la requête externe SELECT NF FROM PUF WHERE NP IN (...) en balayant PUF et en testant pour chaque tuple si ce NP appartient à l'ensemble des numéros de produits rouges.

Dans le WHERE, la condition logique peut avoir plusieurs formes :
- elle peut être composée de conditions élémentaires connectées par AND, OR, NOT, et par des parenthèses;
- les conditions élémentaires sont du type :

<valeur attribut> <opérateur de comparaison> <valeur attribut> |
<valeur attribut> <opérateur d'appartenance> <ensemble>

avec :

<valeur attribut> ::= nom d'attribut | constante
<opérateur de comparaison> ::= > | < | ≥ | ≤ | = | ≠
<opérateur d'appartenance> ::= IN | NOT IN
<ensemble> est soit un ensemble constant, par exemple (1,2,3), soit un ensemble défini par une requête SELECT, soit une union, différence ou intersection d'ensembles :

<ensemble> ::= "(" <constantes> ")" | <requête SELECT> |
<ensemble> <opérateur ensembliste> <ensemble>
<constantes> ::= constante | constante","<constantes>
<opérateur ensembliste> ::= UNION | MINUS | INTERSECT

Exemple : noms des fournisseurs n°1, 2, 3.

```
SELECT    nomF
FROM   F
WHERE NF = 1  OR  NF =2  OR  NF = 3
```

ou

```
SELECT    nomF
FROM   F
WHERE NF  IN  (1, 2, 3)
```

3. Qualification des noms d'attributs

Notations :

NP : attribut
P.NP, PUF.NP : attributs qualifiés par le nom d'une relation

Règle 1 : Un nom d'attribut non qualifié, référence la relation la plus interne qui a un attribut de ce nom-là.

Règle 2 : On peut renommer localement une relation dans la clause FROM .
Exemple: ... FROM PUF PUF1 ...
la relation PUF s'appelle alors PUF1 pour le SELECT correspondant à ce FROM uniquement.

Exemple : noms des fournisseurs ne livrant pas le produit numéro 2.

```
SELECT     nomF
FROM       F
WHERE      2 NOT IN
             (SELECT     NP                  /* ensemble des NP
             FROM        PUF                 livrés par ce
             WHERE       PUF.NF  = F.NF)     fournisseur */
```

Dans le WHERE du SELECT interne, on aurait pu tout aussi bien écrire :

```
WHERE NF = F.NF (cf. règle 1).
```

Exemple : numéros des fournisseurs livrant le même produit que le fournisseur 1 en une quantité plus grande.

Sur le schéma, la requête peut être décrite comme suit :

```
            =1
PUF ( NP, NU, NF, qt )
    =|          |>
PUF ( NP, NU, NF, qt )
```

Ce qui s'écrit en SQL:

```
SELECT  NF
FROM    PUF PUFX                    /* PUF est renommé*/
```

```
WHERE  NP  IN
       (SELECT  NP                          /* ensemble
       FROM  PUF                            des produits du
       WHERE  NF = 1 AND qt < PUFX.qt)      fournisseur 1 */
```

4. Recherche sur plusieurs relations simultanément

Format général:

```
SELECT    Ai...
FROM      R1,  R2..., Rn
WHERE     < condition de jointure entre les Ri >
          AND < condition(s) de la requête >
```

Exemple : pour chaque produit livré, le nom du produit et les villes de leurs fournisseurs.

```
SELECT    nomP, ville
FROM      P, F, PUF
WHERE     PUF.NP = P.NP  AND  PUF.NF = F.NF
```

5. Recherche avec quantificateurs : SOME, ANY, ALL

SQL permet d'écrire des conditions où apparaissent des quantificateurs proches de ceux de la logique ("il existe" (∃), "quelque soit" (∀)), grâce aux mots clefs SOME, ANY et ALL. Les mots clefs SOME et ANY ont exactement la même signification, ce sont des synonymes.

Le format général d'une condition élémentaire avec quantificateur est le suivant:

<valeur attribut> <opérateur de comparaison> <quantificateur> <ensemble>

avec <quantificateur> ::= SOME | ANY | ALL

ce qui signifie:

-.pour SOME et ANY : " existe-t-il dans l'ensemble au moins un élément **e** qui satisfait la condition:
e <opérateur de comparaison> <ensemble> ? "

- pour ALL : " tous les éléments de l'ensemble satisfont-ils la condition ? "

Le mot clef IN est équivalent à un quantificateur existentiel (SOME ou ANY) avec l'opérateur de comparaison d'égalité. SOME et ANY sont donc plus puissants. De même, les requêtes avec un quantificateur universel (ALL) et un comparateur d'égalité peuvent s'écrire avec une condition ensembliste (voir le paragraphe suivant). Cependant le mot clef ALL ne permet pas d'exprimer toutes les requêtes contenant un quantificateur du type "quelque soit". On peut alors écrire la requête inverse avec un "NOT EXISTS" (voir paragraphe plus loin). Par exemple la requête "chercher les X qui pour tout Y satisfont telle condition" peut aussi s'exprimer: "chercher les X tels qu'il n'existe aucun Y qui ne satisfait pas telle condition".

Exemples:

Ensemble des numéros des fournisseurs de produits rouges :

```
SELECT    NF
FROM      PUF
WHERE     NP = ANY
          ( SELECT  NP
            FROM P
            WHERE couleur = "rouge" )
```

Ensemble des numéros des fournisseurs qui ne fournissent que des produits rouges :

```
SELECT    NF
FROM      F
WHERE     "rouge" = ALL
          ( SELECT  couleur
            FROM P
            WHERE NP = ANY     ( SELECT NP   FROM PUF
                                 WHERE PUF.NF = F.NF) )
```

6. Recherche avec des conditions sur des ensembles

Dans les paragraphes précédents, les conditions élémentaires portant sur une valeur d'attribut ont été définies. D'autres types de conditions élémentaires permettent de comparer des ensembles entre eux. Ce sont:

6.1. Test d'égalité d'ensembles:

<ensemble 1> = <ensemble 2>
<ensemble 1> ≠ <ensemble 2>

6.2. Test d'inclusion d'ensembles:

<ensemble 1> CONTAINS <ensemble 2>

Cette condition signifie que l'ensemble 1 contient (ou est égal à) à l'ensemble 2, ce qui en théorie des ensembles s'écrirait : <ensemble 1> ⊇ <ensemble 2>.

La condition:
<ensemble 1> NOT CONTAINS <ensemble 2>
est la négation de la précédente; elle est vraie si un élément de l'ensemble 2 n'appartient pas à l'ensemble 1.

Exemple : noms des fournisseurs qui fournissent tous les produits rouges.

```
SELECT    nomF
FROM      F
WHERE (SELECT  NP
          FROM PUF      /* ensemble des produits du
          WHERE    NF = F. NF) fournisseur F*/
          CONTAINS
          (SELECT    NP
          FROM P        /* ensemble des produits rouges*/
          WHERE     couleur ="rouge")
```

6.3. EXISTS < ensemble>

Cette condition teste si l'ensemble n'est pas vide (ensemble ≠ Ø).

Exemple : noms des fournisseurs qui fournissent au moins un produit rouge.

```
SELECT    nom F
FROM      F
WHERE EXISTS   (SELECT    *
                FROM   PUF, P
          WHERE   NF = F.NF  AND couleur ="rouge"
          AND  PUF.NP=P.NP)
```

Il existe aussi la condition inverse :
NOT EXISTS <ensemble>
qui teste si l'ensemble est vide.

7. Fonctions d'agrégation

SQL offre les fonctions d'agrégation usuelles:

cardinal : COUNT
moyenne : AVG
minimum et maximum : MIN, MAX
total : SUM

qui opèrent sur un ensemble de valeurs prises par un attribut, ou pour COUNT uniquement, sur un ensemble de tuples.

Exemple : quel est le nombre de livraisons faites par le fournisseur 1 ?

```
SELECT   COUNT (*)     /* on compte les tuples de PUF tels
FROM     PUF             que NF = 1 */
WHERE    NF=1
```

Exemple : combien de produits différents a livré le fournisseur 1 ? Attention : il faut ôter les doubles, car COUNT compte toutes les valeurs, y compris les valeurs doubles.

```
SELECT   COUNT (DISTINCT NP)
FROM     PUF
WHERE    NF=1
```

8. Recherche avec partition des tuples d'une relation

8.1. Clause Group by

Exemple: combien de produits différents ont été livrés par chacun des fournisseurs ?
Il faut partitionner l'ensemble des tuples de PUF en un *sous-ensemble (ou groupe)* par numéro de fournisseur. C'est ce que permet la clause GROUP BY.

```
SELECT     NF, COUNT (DISTINCT NP)
FROM       PUF
GROUP BY   NF
```

Cette instruction génère dans le SGBD les actions suivantes :

1. Classer les tuples de PUF, groupe par groupe (un groupe = ensemble des tuples ayant même NF),
2. Pour chaque groupe :
   - évaluer le résultat du SELECT (compter le nombre de NP différents dans ce groupe).

8.2. Clause Having <condition>

Exemple: combien de produits différents ont été livrés par chacun des fournisseurs, tels que la quantité totale de produits livrés par ce fournisseur soit supérieure à 1000 ?

```
SELECT     NF, COUNT (DISTINCT NP)
FROM       PUF
GROUP BY   NF
HAVING     SUM (qt) > 1000
```

La clause "HAVING <condition>" permet de sélectionner les groupes qui satisfont une condition.
Attention, contrairement à la clause "WHERE <condition>", la condition ne porte pas sur un tuple mais sur l'ensemble des tuples d'un groupe.

Le format général du SELECT avec GROUP BY est le suivant :
Soit une relation R (A1, A2, ... , An)

```
SELECT    [Ai1] [,Ai2] ... [,Aiu] [,f1 (Aj1)] [,f2 (Aj2)] .... [,fv (Ajv)]    FROM    R
[ WHERE   <condition1 portant sur chaque tuple de R> ]
```

GROUP BY Ak1 [,Ak2] ... [,Akw]
[HAVING <condition2 portant sur chaque groupe de tuples de R>]

avec fi = fonction d'agrégation (COUNT, SUM, MIN, MAX, AVG),
et {Ak1, Ak2, ... , Akw} ⊇ {Ai1, Ai2, ... , Aiu},
et {Ak1, Ak2, ... , Akw} ∩ {Aj1, Aj2, ... , Ajv} = Ø.

La condition du HAVING peut être de deux types :

1) Condition comparant le résultat d'une fonction d'agrégation portant sur un attribut qui ne fait pas partie de la clause GROUP BY :
f (Aix) <opérateur de comparaison> valeur
Cette fonction porte alors sur l'ensemble des valeurs prises par l'attribut pour le groupe de tuples;

2) Condition comparant l'ensemble des valeurs prises par un attribut (qui ne fait pas partie de la clause GROUP BY) pour les tuples du groupe; cette comparaison se fait en général par rapport à un autre ensemble, à l'aide des comparateurs logiques d'ensembles: =, ≠, CONTAINS, NOT CONTAINS.

Dans ce dernier cas, afin d'exprimer le fait qu'il s'agit de l'ensemble des valeurs prises par l'attribut pour tous les tuples du groupe, le nom de l'attribut est précédé du mot clé SET.
La condition du HAVING peut donc s'écrire :
SET nom-attribut <opérateur de comparaison ensembliste> <ensemble>
avec <opérateur de comparaison ensembliste> ::= = | ≠ | CONTAINS | NOT CONTAINS

L'instruction ci-dessus génère dans le SGBD les actions suivantes :
1. Créer une relation de travail, R', qui est une copie de R; éliminer de R' les tuples qui ne satisfont pas la condition du WHERE.
2. Classer les tuples de R', groupe par groupe (un groupe = ensemble des tuples ayant même Ak1 [,Ak2] ... [,Akw]).
3. Pour chaque groupe de tuples de R' faire:
- tester la condition du HAVING
- si elle est vérifiée, alors évaluer le résultat du SELECT.

Exemple: numéros des fournisseurs qui fournissent au moins tous les produits fournis par le fournisseur numéro 100.

```
SELECT    NF
FROM      PUF
GROUP BY  NF
HAVING    SET NP    /* ensemble des produits du fournisseur NF*/
          CONTAINS
          (SELECT NP FROM PUF WHERE NF=100)
```

9. Mise à jour de la base de données

9.1. Insérer des tuples

- Insérer un tuple : INSERT INTO nomrelation : < tuple constant>
- Insérer un ensemble de tuples :
  INSERT INTO nomrelation :
  <requête SELECT dont le résultat a même schéma que la relation nomrelation>

Exemple : autre catalogue de produits à rajouter à P : Catalogue (NP, nomP, couleur, poids, prix).

```
INSERT INTO  P :
SELECT NP, nomP, couleur, poids
FROM       Catalogue.
```

9.2. Supprimer des tuples

DELETE nomrelation [WHERE condition]

Si la clause "WHERE condition" est présente, seuls les tuples satisfaisant la condition sont supprimés.
Si la clause "WHERE condition" est absente, alors tous les tuples sont supprimés; la relation continue d'exister, mais sa population est vide.

9.3. Mettre à jour un (des) attribut(s)

```
UPDATE    nom relation
SET       nomattr1 = <expression1 qui définit une valeur pour l'attribut1> ,
          .....
          nomattrn = <expression n qui définit une valeur pour l'attribut n>
[WHERE    condition]
```

Exemple : pour les produits verts, changer leur couleur, elle devient "vert d'eau".

```
UPDATE    P
SET       couleur = "vert d'eau"
WHERE     couleur = "vert"
```

10. Plusieurs types d'utilisation de SQL

Le langage SQL a essentiellement été utilisé pour offrir une interface de manipulation d'une base de données à des utilisateurs finaux (non informaticiens), dans un contexte d'interrogation/mise à jour ponctuelle interactive.

Par ailleurs, dans plusieurs systèmes on a développé des passerelles entre SQL et un langage de programmation (C, Cobol, Ada, ...) de façon à permettre l'écriture de programmes qui peuvent ainsi, entre autres, manipuler les données d'une base de données par des instructions SQL.

Enfin, il existe des environnements de programmation, dits de 4ème génération, où un langage de type SQL permet de spécifier les traitements souhaités sur les données, ces spécifications étant ensuite traduites en langage exécutable (invisible à l'utilisateur) par un générateur de programmes.

Ces deux derniers types d'utilisation de SQL (inclus dans un langage de programmation et inclus dans un langage de spécifications type 4ème génération) ont nécessité de développer des mécanismes pour faire cohabiter deux philosophies différentes : celle de SQL dont les instructions "SELECT..." donnent en résultat un ensemble de tuples, et celle des langages type programmation qui travaillent sur un article/tuple à la fois. Pour cela, la solution généralement adoptée est de déclarer le "SELECT...", puis dans une boucle itérative de récupérer, un par un, les tuples du résultat.

Par exemple, pour inclure SQL dans un langage de programmation, la notion de "curseur" est employée. Un curseur est l'équivalent d'un fichier séquentiel temporaire dont le contenu est constitué d'une copie de l'ensemble des tuples résultat d'un *SELECT*.

Exemple de programme en pseudo code + SQL avec curseur :
trouver la liste des noms et villes des fournisseurs du produit de numéro numprod.

```
DECLARE  Fourn  CURSOR FOR
SELECT   nomF, ville
FROM     F
WHERE    NF IN (SELECT NF FROM PUF WHERE NP = numprod) ,
/* déclaration du curseur effectuée */

Lire (numprod) ;     /* entrée du numéro du produit */
OPEN Fourn ;         /* L'OPEN fait exécuter le SELECT du curseur; les tuples résultat sont chargés
                        dans le fichier curseur Fourn */

FETCH Fourn INTO Fnom, Fville;
                     /* un premier tuple est chargé en mémoire centrale
                        Fnom et Fville sont des variables du langage hôte */

Tant que Rep-SGBD = "ok" faire :
                     /* itération pour chaque tuple résultat du curseur */

        Afficher (Fnom, Fville) ;
        FETCH Fourn INTO Fnom, Fville ;      /* le tuple suivant est chargé */

Fin tant que;

CLOSE Fourn ;        /* le curseur est fermé */
```