Chapitre 7

LANGAGE DE MANIPULATION RELATIONNEL :

S Q L

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 SGBD relationnels. Il possède des caractéristiques proches de l'algèbre relationnelle (jointures, opérations ensemblistes) et d'autres proches du calcul des tuples (variables sur les relations). SQL est un langage redondant qui permet souvent d'écrire les requêtes de plusieurs façons différentes. Dans ce chapitre, nous ne présentons que les clauses principales du langage de requêtes et du langage de mise à jour de SQL. Pour une description plus complète, le lecteur est invité à consulter un livre sur SQL.

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 noms 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. La clause ASC signifie selon l'ordre croissant; la clause DESC selon l'ordre décroissant.

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 (∈).
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 (entre autres) du type :

  <valeur attribut> <opérateur de comparaison> <valeur attribut> |

<valeur attribut> <opérateur d'appartenance> <ensemble> |
EXISTS <ensemble> |
NOT EXISTS <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 :
<ensemble> ::= "(" <constantes> ")" | <requête SELECT>
<constantes> ::= constante | constante","<constantes>

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

Ces requêtes permettent d'obtenir dans le résultat (clause SELECT) des attributs provenant de plusieurs relations. Le critère de jointure à effectuer entre les relations peut être défini soit dans la clause WHERE en spécifiant la condition sur les attributs (format 4.1 ci-dessous), soit, si c'est un cas classique de jointure, dans la clause FROM en spécifiant explicitement la jointure entre les relations comme en algèbre relationnelle (format 4.2).

4.1. Recherche sur plusieurs relations avec produit des relations

Format général :

```
SELECT    Aj…
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
```

Si la clause WHERE est absente, alors le résultat de la requête est le produit simple des relations citées dans la clause FROM.

4.2. Recherche sur plusieurs relations avec jointure des relations

Plusieurs formats sont possibles dont les deux suivants :

```
SELECT    Aj…
FROM      R1 JOIN R2 USING Ak, …
[ WHERE … ]
```

Dans ce format l'attribut de jointure Ak doit exister dans R1 et dans R2, et il doit prendre la même valeur pour que les tuples de R1 et de R2 soient joints.

```
SELECT    Aj…
FROM      R1 NATURAL JOIN R2, …
[ WHERE … ]
```

Dans ce format la jointure s'effectue sur tous les attributs communs (de même nom dans les deux relations R1 et R2).

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

```
SELECT    nomP, ville
FROM      PUF JOIN P USING NP JOIN F USING 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. 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 test si un ensemble n'est pas vide : EXISTS

Un format particulier de condition élémentaire est : 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> . Elle teste si l'ensemble est vide.

7. Valeur nulle

Les attributs qui dans le schéma n'ont pas la clause NOT NULL peuvent prendre une valeur particulière, la valeur nulle (NULL). Cette valeur peut être employée dans des cas très différents. Sa sémantique n'est pas définie de façon précise. Par exemple, un attribut à valeur nulle peut signifier que la valeur est (pour le moment) inconnue. Pour un autre attribut, la valeur nulle signifiera que la valeur est inapplicable pour ce tuple (par exemple le nom-de-jeune-fille dans le cas d'un homme). SQL permet d'employer la valeur NULL pour tout attribut, quelque soit son domaine, sauf s'il a été déclaré NOT NULL dans le schéma.

Quand, dans une condition logique élémentaire d'une requête SQL, un attribut de valeur nulle est testé, la valeur logique rendue par la condition n'est ni vrai, ni faux, mais inconnu (UKNOWN). Cela a nécessité d'étendre la logique de Boole à une logique à trois valeurs : TRUE, FALSE et UNKNOWN. Les tables de vérité de cette logique ternaire sont les suivantes.

	NON
vrai	faux
inconnu	inconnu
faux	vrai

OU	**vrai**	**inconnu**	**faux**
vrai	vrai	vrai	vrai
inconnu	vrai	inconnu	inconnu
faux	vrai	inconnu	faux

ET	**vrai**	**inconnu**	**faux**
vrai	vrai	inconnu	faux
inconnu	inconnu	inconnu	faux
faux	faux	faux	faux

Les requêtes SQL rendent en résultat les tuples pour lesquels la condition du WHERE est vraie. Ceux pour lesquels la condition rend inconnu ne font pas partie du résultat.

Si on veut tester si la valeur d'un attribut est nulle, il faut le faire avec la clause particulière IS NULL (ou IS NOT NULL) qui rend toujours vrai ou faux. Employer la condition "attribut = NULL" serait une erreur, car elle rend toujours la valeur logique inconnu.

De la même façon, les opérateurs arithmétiques ont été étendus afin de ne pas générer d'erreur d'exécution quand ils rencontrent une valeur nulle : une opération arithmétique appliquée à une valeur nulle rend NULL.

Cependant, dans de nombreux cas, le traitement des valeurs nulles conduit à des bizarreries. Ce pourquoi, il est fortement conseillé d'éviter les valeurs nulles, par exemple en utilisant les valeurs par défaut. Un exemple de ces bizarreries est le suivant. Soit la relation

Etudiant (n°Etud, nom, prénom, age)

L'union des deux requêtes suivantes :

```
SELECT * FROM Etudiant WHERE age = 22
SELECT * FROM Etudiant WHERE age <> 22
```

ne rend pas en résultat la table Etudiant, car les tuples pour lesquels l'âge a la valeur nulle ne sont sélectionnés par aucune des deux requêtes.

8. 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
```

9. Opérations ensemblistes

SQL permet de faire l'union (UNION), la différence (EXCEPT) et l'intersection (INTERSECT) de relations qui possèdent au moins un attribut de même nom et de domaines compatibles. Lors de ces trois opérations SQL élimine les doubles du résultat (sauf si les variantes UNION ALL, EXCEPT ALL et INTERSECT ALL sont employées).

Exemple : numéros des fournisseurs ayant livré le produit 1 mais jamais le produit 2.

```
(SELECT  NF
FROM  PUF
WHERE  NP = 1)
   EXCEPT
(SELECT  NF
FROM  PUF
WHERE  NP = 2)
```

10. Recherche avec partition des tuples d'une relation : 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 selon l'attribut NF (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).

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 l000 ?

```
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) ,

{Ak1, Ak2, … , Akw} ⊇ {Ai1, Ai2, … , Aiu} ,
{Ak1, Ak2, … , Akw} ∩ {Aj1, Aj2, … , Ajv} = Ø .

La condition du HAVING peut comporter des conditions élémentaires de deux types :
- condition comparant le résultat d'une fonction d'agrégation portant sur un attribut qui ne fait pas partie de la clause GROUP BY :
  fi (Ajx) <opérateur de comparaison> valeur où fi est une fonction d'agrégation
  Cette fonction porte alors sur l'ensemble des valeurs prises par l'attribut pour le groupe de tuples;
- condition portant sur (ou des) attributs de la clause GROUP BY.

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 = un ensemble des tuples ayant même valeur pour Ak1 [,Ak2] … [,Akw]).
3. Pour chaque groupe de tuples de R' faire:
   - tester la condition du HAVING
   - si elle est vérifiée, alors créer un tuple dans le résultat en évaluant la clause SELECT.

Exemple : numéros des fournisseurs qui livrent leurs produits en grande quantité (c'est-à-dire dont la moyenne de leurs quantités livrées est supérieure à 100).

```
SELECT      NF
FROM        PUF
GROUP BY    NF
HAVING      AVG(qt) > 100
```

11. Mise à jour de la base de données

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.
```

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.

Mettre à jour un (des) attribut(s)

```
UPDATE      nom relation
SET         nomattr1 = <expression1 qui définit une valeur pour l'attribut1> ,
             .....
[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"
```

12. 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 */
```