Chapitre IV : LE LANGAGE SQL

Mme Z. TAHAKOURT

1- Introduction

SQL est un langage de définition, de manipulation et de contrôle de données relationnelles.

- Langage de définition des données (LDD) :
  Permet de créer les BD et les tables dans une BD relationnelle, ainsi qu'en modifier et en supprimer.
- Langage de manipulation de données(LMD) :
  Permet de sélectionner, insérer, modifier ou supprimer des données dans une table d'une BD relationnelle.
- Langage de contrôle de données (LCD) :
  Permet de protéger les données des accès non autorisés, il est possible avec SQL de définir des permissions au niveau des utilisateurs d'une BD.

2- Domaines de base

Les domaines utilisés dans les bases de données sont conformes aux types classiquement utilisés dans les langages de programmation, à savoir :

- Entier : INTEGER ;
- Décimal : DECIMAL (m,n) ou NUMBER(mn) où m est le nombre de chiffres et n le nombre de chiffres après la virgule. Ainsi DECIMAL(4,2) peut représenter des chiffres comme 99,99.
- Réel flottant : FLOAT ;
- Chaîne de caractères : CHAR (n) et VARCHAR(n). Les"CHAR" font systématiquement n caractères (les chaînes plus courtes sont complétées par des espaces, les VARCHAR sont taillés au plus juste dans la limite des n caractères ;

3- Base de données exemple

La base de données qui sera utilisée comme de support pour illustrer les requêtes exprimées tout au long de ce cours, est relative à une banque qui désire conserver des infos sur tous ses clients, et aussi sur toutes les agences de la banque mère.

```
Agence(NomAgence, Avoir, Ville)
Clientèle(NClient, nom, prenom, Adresse)
Dépôt(N°Cpt, NClient, NomAgence, Solde)
Crédit(N°Prêt, NClient, NomAgence, Montant)
```

4- Les commandes du LDD

4-1 Création d'un schéma de base de données

Requête1 : création de la base de données *BanqueMere* présentée ci-dessus.

```
CREATE DATABASE BanqueMere
```

4-2 Création d'un schéma de relation

Requête2 : création de la table *Agence* de la BD *BanqueMere*.

```
CREATE TABLE Agence
       (      NomAgence     CHAR(50),
              Avoir  FLOAT,
              Ville  CHAR(20)
       ) ;
```

4-3 Ajout d'un attribut

Requête3 : ajouter l'attribut *NomDirecteur* à la table *Agence*.

```
ALTER TABLE Agence
       ADD COLUMN NomDirecteur CHAR(50);
```

4-4 Suppression d'un schéma de relation

Requête4 : supprimer la table *Clientèle* de *BanqueMere*.

```
DROP TABLE Clientele ;
```

5- Les commandes du LMD

5-1 Langage d'interrogation

Syntaxe générale :

```
SELECT <liste d'attributs projetés>

FROM <liste de relations>

WHERE <liste de critères de restriction et de jointure>
```

Les clauses se renseignent dans l'ordre suivant :

- La partie FROM décrit les relations qui sont utilisables dans la requête (c'est à dire l'ensemble des attributs que l'on peut utiliser). C'est par là que l'on doit commencer par écrire une requête ;
- La partie WHERE exprime la (ou les conditions) que doivent respecter les attributs d'un tuple pour pouvoir être dans la réponse. Cette partie est optionnelle ;
- La partie SELECT indique le sous-ensemble des attributs qui doivent apparaitre dans la réponse (c'est le schéma de la relation résultat). Bien entendu ces attributs doivent appartenir aux relations indiquées dans la partie FROM.

A- Requêtes simples portant sur une seule relation

Les requêtes mono-relation ne concernent par définition qu'une seule relation. Les opérateurs de l'algèbre relationnelle exprimés sont donc la projection et la restriction.

Requête5 : trouver tous les clients de la banque.

En algèbre relationnelle : Clientele
En SQL :

```
SELECT *
FROM Clientele
```

Requête6 : trouver les noms de toutes les agences de la banque mère.

En algèbre relationnelle : $\pi_{\text{NomAgence}}(\text{Agence})$

En SQL :

```
SELECT NomAgence
FROM Agence
```

Requête7 : trouver les numéro des clients de la banque qui habitent la ville de « Brookleen » par ordre croissant de leurs noms.

En algèbre relationnelle : $\pi_{\text{NClient}}[\ \sigma_{\text{Adresse= 'Brookleen'}}(\text{Clientele})]$
Ici on ne peut pas exprimer l'ordre.
En SQL :

```
SELECT NClient
FROM Clientele
WHERE Adresse= 'Brookleen'
ORDER BY nom
```

Remarque syntaxe de « ORDER BY » :

```
ORDER BY <nom d'attribut> [ASC][DESC]
[ASC] : ordre croissant (ordre par défaut)
[DESC]: ordre décroissant
```

Requête8 : trouver les numéro des clients qui ont un compte à l'agence « Perryridge ».

En algèbre relationnelle : $\pi_{NClient}[\ \sigma_{NomAgence= \text{'Perryridge'}}(Depot)]$
En SQL :

```
SELECT NClient
FROM Depot
WHERE NomAgence= 'Perryridge'
```

Requête9 : trouver les numéro des clients qui ont un compte à l'agence « Perryridge » dont le solde dépasse 3000 .

En algèbre relationnelle : $\pi_{NClient}[\ \sigma_{(NomAgence= \text{'Perryridge'}) \text{ et} (Solde>3000)}(Depot)]$
En SQL :

```
SELECT NClient
FROM Depot
WHERE (NomAgence= 'Perryridge')AND(Solde>3000)
```

Requête10 : trouver les noms des agences de la banque dont les noms commencent par 'B' ou 'b' .

En algèbre relationnelle on ne peut pas exprimer le fait que le nom commence par 'B' ou 'b'.

En SQL :

```
SELECT NomAgence
FROM Agence
WHERE (NomAgence LIKE 'B%')OR(NomAgence LIKE 'b%')
```

B- Requêtes portant sur plusieurs relations

Requête11 : trouver les numéros des clients qui ont bénéficié d'un crédit à l'agence « Perryridge » et qui ont un compte dans cette même agence.

En algèbre relationnelle :
$\pi_{NClient}[\ \sigma_{(NomAgence= \text{'Perryridge'})}\ (Depot)] \cup \pi_{NClient}[\ \sigma_{(NomAgence= \text{'Perryridge'})}(Credit)]$

En SQL :

```
SELECT NClient
FROM Depot
WHERE (NomAgence= 'Perryridge')
UNION
SELECT NClient
FROM Credit
WHERE (NomAgence= 'Perryridge')
```

Requête12 : trouver les numéros des clients qui ont un crédit et un compte à l'agence « Perryridge ».

En algèbre relationnelle :

$\pi_{NClient}[\ \sigma_{(NomAgence= 'Perryridge')}\ (Depot)] \cap \pi_{NClient}[\ \sigma_{(NomAgence= 'Perryridge')}\ (Credit)]$

En SQL :

```
SELECT NClient
FROM Depot
WHERE (NomAgence= 'Perryridge')
INTERSECT
SELECT NClient
FROM Credit
WHERE (NomAgence= 'Perryridge')
```

Requête13 : trouver les numéros des clients qui ont un compte à l'agence « Perryridge » mais pas de crédit dans cette même agence.

En algèbre relationnelle :

$\pi_{NClient}[\ \sigma_{(NomAgence= 'Perryridge')}\ (Depot)] - \pi_{NClient}[\ \sigma_{(NomAgence= 'Perryridge')}\ (Credit)]$

En SQL :

```
SELECT NClient
FROM Depot
WHERE (NomAgence= 'Perryridge')
MINUS
SELECT NClient
FROM Credit
WHERE (NomAgence= 'Perryridge')
```

Remarque :
Il faut être très vigilant lors de la manipulation des trois opérateurs ensemblistes l'union, l'intersection et la différence: les deux relations en entrée doivent impérativement être de même schéma.

Formulation de la jointure

Requête14 : trouver les noms des clients avec leurs adresses, qui ont un compte à l'agence « Perryridge ».

En algèbre relationnelle :

$\pi_{\text{nom, Adresse}}[\ \sigma_{(\text{Agence= 'Perryridge'})}\ (\ \text{Depot} \bowtie \text{Clientele})]$

En SQL :

```
SELECT nom, Adresse
FROM Depot, Clientele
WHERE (NomAgence= 'Perryridge') AND (Depot.NClient= Clientele.NClient)
```

Ou,

```
SELECT nom, Adresse
FROM Depot d, Clientele c
WHERE (NomAgence= 'Perryridge') AND (d.NClient=c.NClient)
```

On peut également exprimer la jointure en utilisant l'opérateur IN,

```
SELECT nom, Adresse
FROM Clientele
WHERE (NClient IN (SELECT NClient
        FROM Depot
        WHERE NomAgence= 'Perryridge')
```

Requête15 : trouver les noms, l'adresse et le solde des clients ayant des comptes dans les agences de la ville de « brookleen ».

En algèbre relationnelle :

$\pi_{\text{nom, Adresse, Solde}}\ [\sigma_{(\text{Ville= 'Brookleen'})}\ \text{Agence} \bowtie (\text{Clientele} \bowtie \text{Depot})]$

En SQL :

```
SELECT nom, Adresse, Solde
FROM Agence a, Clientele c, Depot d
WHERE (Ville= 'Brookleen') AND (c.NClient= d.NClient) AND (d.NomAgence=
a.NomAgence)
```

Ou bien,

```
SELECT nom, Adresse, Solde
FROM Clientele c, Depot d
WHERE (c.NClient= d.NClient) AND
      (Agence IN
            (SELECT Agence
             FROM Agence
             WHERE Ville='Brookleen')
      )
```

5-2 Langage de mise à jour

A- Insertion de tuples

Syntaxe:

```
INSERT INTO <NomTable> [Col1, col2,…,Coln]
VALUES (val1, val2,… ,valn)
```

Requête16 : Insérer dans la base un nouveau client : <Ahmed, Tichy>.

```
INSERT INTO Clientele Client, Adresse
VALUES ('Ahmed', 'Tichy')
```

B- Suppression de tuples

Syntaxe:

```
DELETE FROM <NomTable>
[WHERE {CONDITION}]
```

Requête17 : supprimer de la base les agences de la ville de « Brookleen ».

```
DELETE FROM Agence
WHERE Ville=' Brookleen '
```

C- Modification de tuples

Syntaxe:

```
UPDATE <NomTable>
SET col1=nouv_val1, col2=nouv_val2,…, coln=nouv_valn
[WHERE {CONDITION}]
```

Requête18 : Modifier l'adresse du client 'Ahmed' suite à son déménagement, sa nouvelle adresse est 'Toudja'.

```
UPDATE CLIENTELE
SET Adresse='Toudja'
WHERE nom='Ahmed'
```

5-3 Autres formes de requêtes SQL

5-3-1 Les opérateurs IN et NOTIN

Requête19 : trouver les numéro des clients qui ont un compte et un crédit à l'agence « Perryridge ».

```
SELECT NClient
FROM Credit
WHERE (NomAgence='Perryridge') and
(NClient IN (SELECT NClient
             FROM Depot
            WHERE NomAgence='Perryridge')
)
```

Requête20 : trouver les numéro des clients qui ont un compte à l'agence « Perryridge » mais pas de crédit dans cette agence.

```
SELECT NClient
FROM Credit
WHERE (NomAgence='Perryridge') and
(NClient NOTIN (SELECT NClient
             FROM Depot
            WHERE NomAgence='Perryridge')
)
```

5-3-2 L'opérateurs >ANY

Le langage SQL offre la possibilité d'inclure l'opérateur >ANY, ce dernier est utile lorsque se présente dans la requête un segment de phrase tel que : Plus grand que quelconque.

Requête21 : trouver les nom d'agences dont les avoirs sont supérieurs à celui d'une agence de la ville de « Brookleen ».

```
SELECT NomAgence
FROM Agence
WHERE (Avoir >ANY
            (SELECT Avoir
            FROM Avoir
            WHERE Ville='Brookleen'))
```

5-3-3 L'opérateurs >ALL

Cet opérateur est utile lorsque se présente dans la requête un segment de phrase tel que : Plus grand que tous.

Requête22 : trouver les agences dont les avoirs sont supérieurs à ceux de toutes les agences de la ville de « Brookleen ».

```
SELECT NomAgence
FROM Agence
WHERE (Avoir >ALL
            (SELECT Avoir
            FROM Agence
            WHERE Ville='Brookleen')
      )
```

5-3-4 L'opérateurs CONTAINS

Requête23 : trouver les numéro des clients qui ont un compte dans toutes les agences situées dans la ville de « Brookleen ».

En AR : $\pi_{\text{NClient, NomAgence}}(\text{Depot}) / \pi_{\text{NomAgence}}[\sigma_{(\text{Ville= 'Brookleen'})}(\text{Agence})]$

```
SELECT d.NClient
FROM depot d
WHERE (SELECT s.NomAgence
        FROM Depot s
        WHERE s.NClient=d.NClient)

      CONTAINS

      (SELECT NomAgence
      FROM Agence
      WHERE Ville='Brookleen')
```

5-4 Les fonctions d'agrégation

Les agrégats sont des fonctions de calcul. Elles s'appliquent sur l'ensemble des valeurs prises par un attribut et renvoie une valeur unique. Cinq agrégats sont définis : Count, Sum, Avg, Min, Max. La fonction Count(*), un peu particulière, compte les tuples d'une relation.

5-4-1 COUNT

Requête24 : Quel est le nombre de clients de l'agence « Perryridge ».

```
SELECT COUNT(*)
FROM Depot
WHERE NomAgence=' Perryridge '
```

5-4-2 AVG

Requête25 : Quel est l'avoir moyen des agences de la banque.

```
SELECT AVG(Avoir)
FROM Agence
```

5-4-3 SUM

Requête26 : Quel le total des soldes des comptes du client « Ali ».

```
SELECT SUM(Solde)
FROM Depot
WHERE Nclient in (select id from clientele where nom='Ali')
```

5-4-4 Min et Max

Requête27 : Quel est le plus petit (resp. grand) crédit octroyé par l'agence « Perryridge ».

```
SELECT MIN(Montant)
FROM Credit
WHERE NomAgence=« Perryridge ».
```

```
SELECT MAX(Montant)
FROM Credit
WHERE NomAgence=« Perryridge ».
```