Le langage SQL

Sommaire

I. Introduction

L'algèbre relationnelle est la base du développement du langage SQL (Structured Query Langage). Ce langage textuel permet de communiquer avec une base de données relationnelle. Il a été réalisé par IBM puis est devenu un standard ANSI (American National Standards Institute) approuvé par l'ISO (International Standards Organization) en 1987. SQL/92 est la norme très répandue. Le dernier standard validé est SQL3 appelé bien souvent SQL/99. Toutefois, pour avoir un langage plus complet, les concepteurs de SGBD/R ajoutent certaines fonctionnalités au SQL standard.

Parmi les différentes fonctionnalités SQL, il est possible de retrouver les catégories de commandes suivantes :
- DDL (Data Définition Language) qui permet de définir et de modifier le schéma d'une base de données relationnelle.
- DML (Data Manipulation Language) qui permet l'interrogation d'une base de données relationnelle.
- DCL (Data Contrôle Language) pour contrôler la sécurité et l'intégrité de la base de données relationnelle.

Seules l'interrogation et la manipulation des données seront abordées.

II. Le langage de manipulation des données DML

Dans cette première partie on va s'intéresser à la manipulation des données, autrement dit nous allons voir comment interroger nos bases de données (Chercher des données, ajouter des données, modifier des données, supprimer des données)

On distingue typiquement quatre types de commandes SQL de manipulation de données :

- `SELECT` : sélection de données dans une table ;
- `INSERT` : insertion de données dans une table ;
- `DELETE` : suppression de données d'une table ;
- `UPDATE` : mise à jour de données d'une table.

Pour assimiler l'utilisation des différentes requêtes de ce cours, on va utiliser le schéma de la base de données de gestion d'une bibliothèque suivante :

Livre (NumLivre, TitreLivre, AuteurLivre, GenreLivre, EditeurLivre, NombrePages)
Eleve (NumEleve, NomEleve, DateNaissance, LieuNaissance)
Emprunter (NumLivre*, NumEleve*, DateEmprunt, DateRetour)

A. La requête SELECT

La requête SELECT permet de créer des sous-ensembles de données que vous pouvez utiliser pour répondre à des questions spécifiques. Vous pouvez également vous en servir pour fournir des données à d'autres objets de base de données.

La requête SELECT est la requête la plus importante du langage SQL et que nous allons détailler dans ce cours.

Syntaxe de la requête SELECT :

Mots	Arguments
SELECT	Attribut(s)
FROM	Table(s)
WHERE	Condition(s) sur une ligne
GROUP BY	Attribut(s) de partitionnement
HAVING	Condition(s) de sélection sur un groupe de lignes
ORDER BY	Attribut(s) de tri

Les mots clés ne sont pas sensibles à la casse contrairement aux données.

1. Les opérations

a) La projection

Rappel : Une projection permet d'extraire des colonnes spécifiées d'une table.
Une projection s'exprime à l'aide de SQL par la commande :

```
SELECT [DISTINCT] [colonne1 , colonne2 .....]
FROM  nomtable ;
```

Par défaut, SQL n'élimine pas les doubles à moins que cela soit explicitement demandé par le mot clé DISTINCT

Exemple : Pour obtenir la liste des noms des élèves avec leur date de naissance de la table ELEVE il sera nécessaire d'écrire :

```
SELECT NomEleve, DateNaissance
FROM Eleve ;
```

Si l'on souhaite éliminer les doubles, il faudrait noter :

```
SELECT DISTINCT NomEleve, DateNaissance
FROM Eleve;
```

b) La sélection

On parle également à ce niveau de restriction.
Une sélection s'exprime à l'aide de SQL par la commande :

```
SELECT *
FROM  nomtable
WHERE  condition ;
```

Le symbole * signifie que l'on affiche toutes les colonnes d'une table. La combinaison avec une projection s'effectue en remplaçant * par la liste des colonnes à projeter.

Exemple 1 : On désire connaître les caractéristiques du ou des élèves dont le nom est BOUJIDI.

```
SELECT *
FROM Eleve
WHERE  NomEleve = 'BOUJIDI';
```

La même requête avec l'affichage du NumEleve et de la date de naissance s'écrit :

```
SELECT NumEleve, DateNaissance
FROM ELEVE
WHERE  NomEleve = 'BOUJIDI';
```

De manière plus générale, la condition suivant la commande WHERE peut inclure différents opérateurs de comparaison et différents opérateurs booléens :

***** les opérateurs de comparaison :

SYMBOLE	SIGNIFICATION
=	Égal
< > !=	Différent
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

***** les opérateurs logiques :

SYMBOLE	SIGNIFICATION
Between...AND...	Entre ... et ...
Like	Comme
IS NULL	Dont la valeur est nulle
IN	Compris dans la liste

Avec l'opérateur Like, il est possible d'utiliser des caractères "joker" comme % et _.

Le "joker" % remplace de 0 à n caractères quelconques.
Exemple :
Si l'on écrit après la commande **WHERE attribut Like 'F%'**, la condition portera sur toutes les valeurs de l'attribut dont la première lettre commence par F.

Le "joker" _ remplace un caractère quelconque et un seul.
Exemple : Si l'on écrit après la commande **WHERE attribut Like'F_ _'** , la condition portera sur toutes les valeurs de l'attribut dont la première lettre commence par F et ayant ensuite seulement 2 autres caractères.

*** les opérateurs booléens :**

SYMBOLE	SIGNIFICATION
AND	ET
OR	OU
NOT	NON

L'opérateur NOT inverse la valeur du résultat. Il est ainsi possible d'écrire : NOT BETWEEN, NOT IN, NOT LIKE, IS NOT NULL,

À ces opérateurs booléens, il peut être ajouté des parenthèses pour indiquer l'ordre d'évaluation.

***le tri :**

Il est possible de trier les résultats suivant l'ordre ascendant (mot clé : ASC) ou descendant (mot clé : DESC) d'une ou plusieurs colonnes.

La commande SQL sera la suivante :
ORDER BY {expression | position} [ASC | DESC] [,{expression | position} [ASC | DESC] ....]

Exemple : on désire relever les élèves dont le nom est BOUJIDI en affichant le numéro d'élève et le lieu de naissance. Les données seront présentées dans l'ordre croissant des lieux de naissance. L'instruction SQL sera la suivante :

```
SELECT NumEleve, LieuNaissance
FROM Eleve
WHERE  NomEleve ='BOUJIDI'
ORDER BY LieuNaissance ;
```

Si l'on veut obtenir les mêmes données mais en affichant les lieux de naissance dans un ordre décroissant, l'instruction SQL peut être la suivante :

```
SELECT NumEleve, LieuNaissance
FROM Eleve
WHERE  NomEleve='BOUJIDI'
ORDER BY LieuNaissance DESC ;
```

c) La jointure

Cette opération consiste à joindre 2 tables avec un critère de sélection. Une sélection s'exprime à l'aide de SQL par la commande :

```
SELECT *
FROM  nomtable1, nomtable2
WHERE  nomtable1.clé=nomtable2.clé ;
```

Exemple : on veut joindre la table EMPRUNTER et la table ELEVE.
L'instruction SQL sera la suivante :

```
SELECT *
FROM  Eleve, Emprunter
WHERE  Eleve.NumEleve=Emprunter.NumEleve ;
```

Il est possible de renommer les tables par des alias. Cela permet notamment d'écrire plus rapidement le code pour réaliser des requêtes et de réduire les erreurs de frappe. Ainsi pour l'exemple précédent, il est possible d'écrire la même requête de la manière suivante :

```
SELECT *
FROM  Eleve C , Emprunter  E
WHERE  C.NumEleve=E.NumEleve ;
```

Avec le langage SQL, il est possible de combiner les différentes opérations, il est ainsi possible de donner la liste des livres (titre des livres, genre et éditeur) empruntés par les élèves avec le nom 'RADI'. Dans la table on fera également apparaître le NumEleve pour distinguer éventuellement les élèves ayant le même nom.

Les différentes instructions sont les suivantes :

```
SELECT C.NumEleve, TitreLivre, GenreLivre, EditeurLivre
FROM Eleve C, Emprunter E, Livre L
WHERE NomEleve='RADI'
AND C.NumEleve = E.NumEleve
AND E.NumLivre = L.NumLivre ;
```

d) Les fonctions de calcul et les regroupements

***** Les fonctions de calcul

Il est possible d'effectuer dans une requête SQL, des calculs horizontaux sur des lignes en utilisant les opérateurs +, -, *, / aussi bien dans la commande SELECT que dans la commande WHERE. Les arguments des opérateurs sont des noms de colonnes de type numérique ou des constantes.

Exemple : on souhaite afficher la table EMPRUNTER avec des dates d'emprunte décalées de 2 jours. L'instruction SQL sera la suivante :

```
SELECT NumEleve, CodeLivre, DateEmprunt + 2
FROM  Emprunter ;
```

Exemple : on souhaite afficher le numéro des élèves qui ont rendu le livre après 20 jours d'emprunt.

```
SELECT NumEleve
FROM  Emprunter
WHERE DateRetour > DateEmprunt +20 ;
```

*** Les fonctions agrégatives**

Elles permettent d'effectuer des calculs verticaux pour l'ensemble ou un sous-ensemble des valeurs d'une colonne.
Les fonctions principales sont les suivantes :

FONCTIONS	SYMBOLE
SUM	permet d'effectuer la somme des valeurs d'une colonne numérique
AVG	permet d'effectuer la moyenne des valeurs d'une colonne numérique
MAX	permet de rechercher la valeur maximale d'une colonne numérique
MIN	permet de rechercher la valeur minimale d'une colonne numérique
COUNT	permet de compter le nombre de valeurs d'une colonne

- Pour compter le nombre de lignes sélectionnées, la fonction COUNT doit être utilisée avec l'argument * (COUNT(*)).
- Pour compter le nombre de valeurs distinctes prises par une colonne, il faut indiquer l'argument DISTINCT suivi de l'argument considéré.

- Les fonction agrégatives dans des instructions portant sur l'ensemble d'une colonne

Il n'y a pas ici de difficultés particulières

Exemple 1 :
Vous souhaitez déterminer le nombre de Eleves.

```
SELECT COUNT(NumEleve) AS NbEleve
FROM ELEVE ;
```

Exemple 2 :
Vous cherchez à déterminer le total des pages de tous les livres.

```
SELECT SUM(NombrePages) AS SommePages
FROM Livre
```

Exemple 3:
On désire déterminer la moyenne des pages pour les livres. La requête sera la suivante :

```
SELECT AVG(NbrePages) AS MoyPage
FROM Livre ;
```

- Les fonction agrégatives dans des Instructions portant sur les sous-ensembles d'une colonne

° Le partitionnement

Il doit permettre d'effectuer des calculs statistiques pour chaque sous-ensemble de lignes vérifiant un même critère. Le partitionnement s'exprime en SQL par la commande GROUP BY suivie du nom des colonnes de partitionnement.

Exemple 1 :
Vous souhaitez déterminer le nombre d'emprunt pour chaque Eleve ayant emprunté un livre.

```
SELECT NumEleve, COUNT(NumEleve)
FROM EMPRUNTER
GROUP BY NumEleve ;
```

Exemple 2 :
Vous cherchez à déterminer le total des pages par éditeur

```
SELECT EditeurLivre, SUM(NombrePages)
FROM LIVRE
GROUP BY EditeurLivre ;
```

Exemple 3 :
Vous cherchez à déterminer la moyenne du nombre moyen de pages par éditeur.

```
SELECT EditeurLivre, AVG(NombrePages)
FROM LIVRE
GROUP BY EditeurLivre;
```

° Les conditions sur des classes de lignes.

Lorsqu'une condition de recherche doit être exprimée non pas sur les lignes (WHERE) d'une table mais sur les **classes de lignes** introduites par un partitionnement, il est nécessaire d'utiliser la commande **HAVING**.

Exemple : Vous cherchez à sélectionner et visualiser pour chaque éditeur le nombre moyen de pages proposées. Seuls les éditeurs proposant une moyenne du nombre de pages supérieure à 83,1 devront être visualisés.

```
SELECT EditeurLivre, AVG(NombrePages) as MoyPages
FROM LIVRE
GROUP BY EditeurLivre
HAVING MoyPages >83.1 ;
```

2. Les requêtes imbriquées

Elles sont appelées également sous-requêtes. Il s'agit d'une requête incorporée dans la commande WHERE ou HAVING d'une autre requête (requête principale). Cette dernière utilise les résultats de la sous-requête.
Certaines sous-requêtes permettent de remplacer les jointures. Les sous-requêtes renvoient une ou plusieurs valeurs.

La requête ne renvoie qu'une seule valeur.

L'imbrication de la requête avec la sous-requête se fera avec un opérateur de comparaison (=, >, <=, …)

Exemple : Quel est le CodeLivre ayant le nombre de pages le plus élevé ?

```
SELECT CodeLivre
FROM Livre
WHERE
NombrePages = (SELECT Max(NombrePages)  FROM Livre );
```

Remarque : La sous-requête est notée entre parenthèses.

2- La sous-requête renvoie plusieurs valeurs

L'imbrication se fera avec bien souvent l'opérateur logique IN

Exemple :
On souhaite connaître la liste des numéro des élèves qui ont emprunté un livre contenant plus que 120 pages

La requête SQL peut alors être la suivante :

```
SELECT NumEleve
FROM Emprunter, Livre
WHERE
Emprunter.NumLivre=Livre.NumLivre   AND   NombrePages>=120
```

Elle peut également être la suivante :

```
SELECT NumEleve
FROM EDITEUR
WHERE
NumLivre IN  (SELECT NumLivre FROM  Livre WHERE NombrePages>=120)
```

Dans la première formulation, la mention DISTINCT permet d'éliminer les doublons.
ORDER BY ne peut être noté dans une sous-requête.

B. La commande INSERT

Elle permet d'insérer un ou plusieurs tuples dans une table. Il est possible de dire également que cette commande va permettre d'assurer le "peuplement" des tables de la base à partir de données nouvelles.

Le formalisme est le suivant :

```
INSERT INTO <TABLE>
VALUES ( ' valeur1 ',' valeur2 ') ;
```

Exemple :

```
INSERT INTO  EDITEUR
VALUES ('LACO1','BertrandLacopee', '8, rue de Nevers','75009','PARIS')
```

Si la mise à jour n'existe que pour quelques attributs, le formalisme est le suivant :

```
INSERT INTO <TABLE> (ATTRIBUT1, ATTRIBUT2)
VALUES ('valeur1','valeur2') ;
```

Il est alors nécessaire de respecter l'ordre des attributs.

Exemple :
On souhaite ajouter un livre. Deux éléments seulement sont connus le codelivre (F132) et le codeediteur (FOU1)

```
INSERT INTO LIVRE (CodeLivre, CodeEditeur)
VALUES ('F132','FOU1')
```

Dans tous les cas, les valeurs de la liste sont séparées par des virgules. Des guillemets simples doivent encadrer les données dès qu'elles comportent des caractères ou des dates. C'est inutile pour des données numériques ou pour des valeurs NULL (absence de valeurs).
Ainsi pour l'exemple précédent, il était également possible d'écrire :

```
INSERT INTO LIVRE
VALUES ('F132','','','FOU1')
```

Ou bien encore, il était possible d'écrire :

```
INSERT INTO LIVRE
VALUES ('F132',NULL,NULL,FOU1)
```

Rque : L'absence de valeur pour un attribut (ou colonne) doit avoir été autorisée. C'est par exemple impossible pour la clé primaire.

C. La commande UPDATE

Elle permet de mettre à jour des données dans une table.
Le formalisme est le suivant :

```
UPDATE <table>
SET <attribut>=<' valeur '>
[WHERE <condition>] ;
```

Si le mot réservé WHERE est facultatif et permet de faire une mise à jour sous certaines conditions, il est très rare de ne pas le noter. L'inverse peut-être dangereux car l'ensemble de la colonne serait modifié.

Exemple : Le code éditeur a maintenant pour nom Foucher et Compagnie.

```
UPDATE EDITEUR
SET NomEditeur = 'Foucher et Compagnie'
WHERE CodeEditeur = 'FOU1'
```

S'il y a 2 mises à jour dans la même requête, il faut les séparer par une virgule et ne pas mettre 2 fois SET
La syntaxe est la suivante :

```
UPDATE <table>
SET     <attribut1>=<' valeur2 '>,
        <attribut2>=<' valeur4 '>
[WHERE <condition>] ;
```

Exemple : Le code éditeur a maintenant pour nom Fourcher et Compagnie et pour adresse 31 rue de Fleurus.

```
UPDATE EDITEUR
SET     NomEditeur = 'Fourcher et Compagnie'
        AdresseEditeur ='31, rue de Fleurus'
WHERE CodeEditeur = 'FOU1'
```

D. La commande DELETE

Elle permet d'effacer un ou plusieurs tuples.
DELETE FROM <table>
[WHERE <condition>] ;
Ici aussi le mot réservé WHERE est facultatif et permet de faire une suppression sous certaines conditions. Mais de la même manière, il est très rare de ne pas le noter. L'inverse peut-être dangereux car l'ensemble des enregistrements serait supprimé.

Exemple
On souhaite supprimer l'éditeur dont le code est F132.
La requête SQL est la suivante :

```
DELETE FROM EDITEUR
WHERE CodeEditeur IN('F132')
```