Chapitre VI : Le langage SQL

I. Historique

Le langage SQL (Structured Query Language) est un langage d'interrogation structurée de données. C'est un langage complet de gestion de bases de données relationnelles. Il a été conçu par IBM dans les années 70 et est devenu le langage standard des systèmes de gestion de bases de données relationnelles. Il existe plusieurs versions de SQL dont :

- **SQL 86 :** Normalisé en 1986 par l'ANSI (American National Standards Institute),
- **SQL 89 :** Approuvé par ISO (International Organization for Standardization) et ANSI,
- **SQL 92 :** Appelé **SQL 2** est un standard sorti en 1992 et approuvé par ISO et ANSI,
- **SQL 3 :** Sorti en 1999, est une extension du standard 92 et prend en compte la récursion, les déclencheurs (triggers), etc.

Le langage SQL est divisé en trois grandes parties que sont :

1. **Le Langage de Définition de Données (LDD) :** Il correspond aux commandes de SQL qui permettent de créer, modifier ou supprimer la définition d'une base de données, d'une table, d'une vue ou d'autres objets.
2. **Le Langage de Manipulation de Données (LMD) :** Il permet de manipuler le contenu d'une base (affichage) ou de faire des mises à jour (ajout, modification ou suppression d'enregistrements).
3. **Le Langage de Contrôle de Données (LCD) :** Il permet de contrôler l'accès à l'information contenue dans la base avec la création d'utilisateurs, mais aussi l'attribution de droits d'accès appelés privilèges à ces utilisateurs.

Remarque : Le langage SQL n'est pas sensible à la casse. Toutefois, cette insensibilité à la casse n'est que partielle dans la mesure où la différenciation entre minuscules et majuscules existe au niveau des identificateurs d'objets.

II. Les types

Pour chaque attribut d'une table, on doit préciser l'ensemble des valeurs que les enregistrements peuvent prendre. Le langage SQL prend en compte les types de données suivants :

II. 1. Les types numériques

smallint : Entiers codés sur 2 octets, les valeurs possibles sont comprises entre **-2^{16-1} (-32768)** et **$2^{16-1} - 1$ (32767)** ;

integer : Entiers codés sur 4 octets. Les valeurs vont de **-2^{32-1} (-2147483648)** à **$2^{32-1} - 1$ (2147483647)** ;

bigint : Entiers codés sur 8 octets. Les valeurs vont de **-2^{64-1}** à **$2^{64-1} - 1$** ;

decimal(k[, n]) : Réels dont le nombre total de chiffres (partie entière et décimale comprises) est **k** et le nombre de chiffres de la partie décimale est **n** et est optionnel. S'il n'est pas donné, il est supposé nul.

II. 2. Les types alphanumériques

char(n) : C'est des chaines de n caractères. Leur longueur est fixe ;

varchar(n) : C'est des chaines de caractères dont la longueur maximum est n.

II. 3. Les types horaires

date : Champ de type date sous la forme AAAA-MM-JJ ;

time : Champ de type heure sous la forme HH:MM:SS:TT ;

timestamp : Champ de type Date et heure simultanément.

II. 4. Autres types

bit(n) : Succession de bits (des 0 et des 1) de longueur n ;

boolean : Valeurs booléennes (Vrai et Faux) (Si le SGBD supporte SQL 3) ;

decimal(k[, n]) : Il fonctionne de la même manière que le type ***number***.

Remarque :

Il est possible d'ajouter des commentaires dans un code SQL.

- A la suite du caractère % : Commentaires sur une ligne ;
- Entre /* et */ : Commentaires sur plusieurs lignes.

III. Langage de Définition de Données (DDL)

C'est la partie du langage SQL qui permet de créer la base, ses tables, ses vues, etc.

III. 1. Création de tables

Une table est la représentation physique d'une relation. Elle est représentée par un tableau dont les colonnes sont les attributs de la relation et les lignes sont les enregistrements de l'instance de la relation. La syntaxe de la création d'une table est :

Create Table Nom_Table
(Attribut_1 Type_Attribut_1 [Contrainte sur la colonne],
- - - - - - - -
Attribut_n Type_Attribut_n [Contrainte sur la colonne],
[Contrainte sur la table],
- - - - - - - -
[Contrainte sur la table]
) ;

Exemple :

Create Table Pays
(Nom varchar(25),
Devise varchar(50),
Hymne varchar(30),
Superficie decimal(12, 3),
Population integer
) ;

III. 1. 1. Les contraintes

Le langage SQL permet de définir deux types de contraintes que sont les contraintes de colonnes et les contraintes de tables.

III. 1. 1. 1. Contraintes de colonnes

a. Clé primaire : Attribut Type **Primary Key**,

Cette méthode ne peut être utilisée que si la clé ne comporte qu'un seul attribut

b. Clé étrangère : Attribut Type **Foreign key References** TableRef(AttributRef),

c. Valeur Nulle : Attribut Type **Not Null**,

d. Valeur par défaut : Attribut Type **Default** Valeur,

e. Valeur unique : Attribut Type **Unique**,

f. Restriction sur un type de base :

Attribut Type **Check**(Attribut **Between** Valmin and Valmax),

Attribut Type **Check**(Valuet **In** (Val_1, Val_2, …, Val_n)),

Attribut Type **Check**(Attribut **Like** 'Format'),

Remarque : ***TableRef*** et ***AttributRef*** désignent respectivement la table dans laquelle se trouve la clé primaire référencée et l'attribut qui est la clé primaire de cette table.

Exemple : On suppose que :

1. L'attribut ***Nom*** est la clé primaire de la table Pays ;
2. Deux pays ne peuvent pas avoir le même hymne national ;
3. La superficie d'un pays est obligatoire ;
4. L'attribut President est une clé étrangère qui référence la clé primaire NumId d'une table Patriote.
5. La population des pays considérés est entre 1000000 et 15000000

Create Table Pays

```
(   Nom          varchar(25)      Primaty Key,
    Devise       varchar(50),
    Hymne        varchar(30)      Unique,
    Superficie   decimal(12, 3)   Not Null,
    Population   integer          Check(Value Between 1000000 and 15000000),
    President    varchar(15)      Foreign Key References Patriote(NumId)
) ;
```

III. 1. 1. 2. Contraintes de table

Il est possible de faire des contraintes données ci-dessus des contraintes de tables. La syntaxe est :

Create Table Nom_Table

```
(   Att_1                 Type_Att_1,
    - - -  -  -  -  -
    Att_P                 Type_Att_P,
    Att_P+1               Type_Att_P+1,
     - - -  -  -  -  -
    Att_n-1               Type_Att_n-1,
    Att_n                 Type_Att_n,
    Constraint Nom Primary Key (Att_1, …, Att_P),
    Constraint Nom Foreign Key (Att_n-1, Att_n) References TabRef(AttRef1, AttRef2),
    Constraint Nom Not Null (Attribut),
    Constraint Nom Check(Attribut In (Val_1, Val_2, …, Val_n))
) ;
```

Create Table Pays

```
(  Nom          varchar(25),
   Devise       varchar(50),
   Hymne        varchar(30),
   Superficie   decimal(12, 3),
   Population   integer,
   President    varchar(15),
   Constraint   Pk_Pays      Primary Key (Nom),
   Constraint   Fk_PaysPat Foreign Key (President) References Patriote(NumId),
   Constraint   NN_SupPays Not Null (Superficie),
   Constraint   Ck_PopPays Check(Population Between 1000000 and 15000000),
   Constraint   Uq_HymnePays Unique(Hymne)
) ;
```

Exemple :

Etudiant (Num_Carte, Nom, #FiliereE)

Filiere (NomF, Responsable, Adresse)

Create Table Filiere

```
(  NomF          varchar(40),
   Responsable   char(15) Unique,
   Adresse       varchar(40),
   Constraint PK_Filiere Primary Key (NomF)
) ;
```

Create Table Etudiant

```
(  Num_Carte     integer,
   Nom           varchar(35),
   FiliereE      varchar(40),
   Constraint PK_Etudiant Primary Key (Num_Carte),
   Constraint FK_Etudiant_Filière Foreign Key (FiliereE) References Filiere (NomF),
   Constraint NN_NomEtu Not Null (Nom),
   Constraint CK_FiliereEtu Check(Filiere In ('MPI', 'MP', 'PC'))
) ;
```

Remarques :

- **Default :** Elle permet de donner une valeur par défaut lors de l'insertion, si une valeur explicite n'est pas donnée pour cette colonne ;
- **Unique :** Elle permet de spécifier que pour cette colonne, deux enregistrements ne peuvent pas avoir la même valeur ;
- **Check :** Elle permet de faire des restrictions supplémentaires sur les types de base des attributs.

Ainsi dans les tables ci-dessus (**Filiere** et **Etudiant**) :

- Une personne ne peut pas être responsable de deux filières différentes (**Unique**) ;
- Le nom d'un étudiant est obligatoire (**Not Null**) ;
- La filière d'un étudiant est soit MPI, soit MP soit PC (**Check**) ;
- Les attributs **Num_Carte** et **NomF** sont respectivement les clés primaires des tables **Etudiant** et **Filiere** ;
- L'attribut **FiliereE** de la table **Etudiant** est une clé étrangère qui référence la clé primaire (**NomF**) de la table **Filiere**.

III. 1. 2. Les domaines

On peut créer nos propres types de données à partir des types préexistants en utilisant la notion de **domaine**. Un domaine est créé comme suit :

Create Domain Nom_Domaine **AS** Type_De_Donnee
[**Default** Valeur_Par_Defaut] Contrainte ;

Exemple : On crée deux domaines **Note** et **Jour** tels que :

1. Le domaine **Note** contient les valeurs réelles comprises entre 0 et 20 écrites avec 4 chiffres dont 2 après la virgule:

   Create Domain Note **As** Number (4, 2)
   Default 0 **Check** Value **between** 0 **and** 20 ;

2. Le domaine **Jour** Contient les chaines de caractères réduites aux 7 jours de la semaine :

   Create Domain Jour **As** Varchar(8)
   Check Value **In** ('Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi', 'Dimanche') **Not Null** ;

Une fois créé, le domaine peut être utilisé comme les autres types.

Exemple :

Create Table Cours

(Id char(5) **Primary Key**,
HeureDebut time,
HeureFin time,
Journee **Jour**
) ;

On peut modifier un domaine comme suit :

Alter Domain Nom_Domaine [Add] | [Modify] Contrainte ;

III. 2. Création d'une base de données

Une base de données est créée avec la syntaxe suivante :

Create Database Nom_Base ;

??????????????????????????????????????

III. 3. Modification de la structure d'une table

Pour modifier la structure d'une table, on utilise la commande **Alter Table**, et pour supprimer une table, on utilise la commande **Drop Table**.

a. Ajouter un attribut : Un attribut est ajouté dans une table avec la clause **Add**.

Syntaxe : Alter Table Nom_Table **Add** Att_1 type_1, ….., Att_m type_m ;

b. Renommer un attribut : Il est possible de renommer un attribut et changer sont type en même temps avec la clause **Change**.

Syntaxe : Alter Table Nom_Table **Change** Ancien_Nom Nouveau_Nom Nouveau_type ;

c. Supprimer un attribut : Un attribut est supprimé avec la clause **Drop**

Syntaxe : Alter Table Nom_Table **Drop** Nom_Attribut ;

d. Modifier le type d'un attribut : Il est possible de :

- augmenter ou diminuer la taille d'un attribut ;
- changer le type d'un attribut vide ;

Un attribut est modifié avec la clause **Modify**.

Syntaxe : Alter Table Nom_Table **Modify** (Attribut type) ;

e. Ajouter une contrainte : Il est possible d'ajouter une contrainte (clé primaire, clé étrangère, Not Null, Unique, Check, Default) sur un attribut d'une table déjà créée.

Syntaxe : Alter Table Nom_Table **Add Constraint** Nom_Contrainte Contrainte ;

f. Supprimer une contrainte : Une contrainte est supprimée avec la clause **Drop**.

Syntaxe : Alter Table Nom_Table **Drop** Primary Key ;

Syntaxe : Alter Table Nom_Table **Drop** Constraint Nom_Contrainte ; /* Contraintes nommées */

g. Suppression d'une table : Une table est supprimée par la clause **Drop Table**

Syntaxe : Drop Table Nom_Table ;

h. Renommer une table : On peut renommer une table en utilisant la clause **Rename**.

Syntaxe : Rename Table Ancien_nom **TO** Nouveau_nom ;

Exemple : On reprend les tables de l'exemple précédent et on ajoute la colonne **Adresse** à la table **Etudiant**, on modifie le domaine de la colonne **Responsable** de la table **Filiere**, on supprime la contrainte **Not Null** sur la colonne **Nom** de la table **Etudiant** et.

1. **Alter Table** Etudiant **Add** Adresse varchar(30) ;
2. **Alter Table** Filiere **Modify** (Responsable varchar(30)) ;
3. **Alter Table** Filiere **Drop** Adresse ;
4. **Alter Table** Etudiant **Drop Constraint** NN_NomEtu ;

III. 4. Les vues

Une vue est une partie d'une table dont la structure est décrite dans une requête portant sur la table en question. Elle permet de restreindre l'accès d'une table à une partie de son contenu pour certains utilisateurs. Ainsi, chaque utilisateur peut avoir sa propre vision des données. Elle peut contenir une partie des colonnes de la table, une partie de ses enregistrements, ou les deux en même temps. Les données ne sont pas dupliquées, seule la définition de la vue est stockée.

Création d'une vues : La création d'une vue est faite avec la clause **Create View**

Create view Nom_Vue [(Att_1, Att_2, …, Att_n)] **As Select** Requête ;

Remarque : La liste des attributs n'est pas obligatoire. Si elle est omise, les attributs de la vue auront même nom que ceux de la table sur laquelle elle est extraite. Cependant, s'il y a

des attributs résultats de la requête qui sont des expressions, on est obligé de les renommer ou de citer les noms des colonnes de la vue lors de sa création.

On peut ajouter **Check Option** à la fin de la définition de la vue pour interdire la mise à jour ou l'insertion d'enregistrements qui ne répondent pas au(x) critère(s) de la requête.

Une vue est modifiée de la même manière que toutes les autres tables. Cependant, elle ne peut être mise à jour (**Insert**, **Update**, **Delete**) que si elle obéît à un certain nombre de conditions :

- Ne porter que sur une table (pas de jointure) ;
- Ne pas contenir de dédoublonnage (pas de mot clé **Distinct**) si la table n'a pas de clé ;
- Contenir la clé de la table si elle en a ;
- Ne pas transformer les données (pas de concaténation, addition de colonne, calcul d'agrégat) ;
- Ne pas contenir de clause **Group By** ou **Having** ;
- Ne pas contenir de sous-requête ;
- Répondre au filtre **Where** si la clause **With Check Option** est spécifiée lors de la création de la vue.

Supprimer une vue : Pour supprimer une vue, on utilise la clause **Drop View**.

Drop View Nom_Vue ;

Renommer une vue : Pour renommer une vue, on utilise la clause **Rename**.

Rename View ancien_nom **TO** nouveau_nom ;

Interrogation des vues : Les vues sont interrogées de la même manière que les autres tables avec la commande **Select**.

Exemple :

1. Création d'une vue sans changer les noms des attributs :

Create View EtudiantZig **As Select** * **From** Etudiant **Where** Adresse = 'Ziguinchor' ;

2. Création d'une vue en changeant les noms des attributs :

Create View EtudiantZig (Num_CarteZig, NomZig, FiliereZIG, AdresseZig)

As Select * **From** Etudiant **Where** Adresse = 'Ziguinchor' ;

IV. Langage de Manipulation de Données (DML)

Le langage de manipulation de données permet de mettre à jour une base de données. Elle permet aussi de l'interroger pour consulter son contenu ou extraire des informations pour une utilisation spécifique.

IV. 1. Les opérateurs et expressions de restriction

Une restriction consiste à sélectionner des enregistrements selon une condition logique appliquée sur les attributs d'une relation. En SQL, les restrictions s'expriment à l'aide des clauses **Where** et/ou **Having** suivies d'une condition logique exprimée à l'aide des éléments suivants :

- **Opérateurs logiques :** And, Or, Not ;
- **Comparateurs de chaînes :** In, Between, Like ;
- **Opérateurs arithmétiques :** +, -, *, / ;
- **Comparateurs arithmétiques :** =, >, <, >=, <=, <> ;
- **Autres opérateurs :** All, Any, Exist.

Remarque :

Le comparateur **Like** utilise des jokers pour remplacer des valeurs quelconques. Ces jokers sont donnés dans le tableau suivant :

Jokers	Signification
? ou _	Remplace un caractère unique
* ou %	Remplace une suite de caractères
#	Remplace un chiffre unique
[v_1 - v_2] ou [v_1, v_2, v_3]	Donne une plage ou liste de caractères
[!] ou [^]	Interdit un ou plusieurs caractères

IV. 2. La commande SELECT

Elle permet de chercher une information dans une multitude d'informations stockées dans une base de données. En SQL toutes les opérations de l'algèbre relationnelle (Sélection, Projection, Jointure, etc.) commencent par le mot clé **Select**.

IV. 2. 1. La sélection

La syntaxe de la sélection est :

Select * From Nom_Table **Where** Condition(s) ;

Exemple : Supposons qu'on ait la table **Etudiant** suivante :

Etudiant

NumCarte	Nom	Prénom	Niveau	Promo
C001	Diatta	Souléymane	L1	1
B002	Gueye	Mamadou	L2	1
D001	Seck	Fatoumata	M1	2
E004	Traoré	Souléymane	L1	2

1. **Select** * **from** Etudiant **Where** (Etudiant.Niveau = 'L1') Or (Etudiant.Niveau = 'M1') ;

NumCarte	Nom	Prénom	Niveau	Promo
C001	Diatta	Souléymane	L1	1
D001	Seck	Fatoumata	M1	2
E004	Traoré	Souléymane	L1	2

Cette requête donne tous les attributs des étudiants de la L1 et de la M1.

2. **Select** * **From** Etudiant **Where** Etudiant.Prénom = 'Souléymane' ;

NumCarte	Nom	Prénom	Niveau	Promo
C001	Diatta	Souléymane	L1	1
E004	Traoré	Souléymane	L1	2

Cette requête donne tous les attributs des étudiants nommés Souléymane.

IV. 2. 2. La projection

La syntaxe de la projection est :

Select liste, attributs **From** Nom_Table ;

Exemple : Avec la même table l'exécution des requêtes suivantes donne :

1. **Select** Etudiant.NumCarte, Etudiant.Promo **From** Etudiant ;

NumCarte	Promo
C001	1
B002	1
D001	2
E004	2

Cette requête donne le numéro de carte et la promotion de tous les étudiants.

2. **Select** Etudiant.Nom, Etudiant.Prénom **From** Etudiant ;

Nom	Prénom
Diatta	Souléymane
Gueye	Mamadou
Seck	Fatoumata
Traoré	Souléymane

Cette requête donne le nom et le prénom de tous les étudiants.

IV. 2. 3. Sélection et Projection combinées

La syntaxe d'une sélection combinée avec une projection est :

Select liste_d'attributs **From** Nom_Table **Where** Condition(s) ;

Exemple : On considère toujours la même table **Etudiant** :

1. **Select** Etudiant.NumCarte, Etudiant.Nom, Etudiant.Prénom **From** Etudiant **Where** Etudiant.Promo = 1 ;

NumCarte	Nom	Prénom
C001	Diatta	Souléymane
B002	Gueye	Mamadou

Cette requête donne le numéro de carte, le nom et le prénom des étudiants de la première promotion.

2. **Select** Etudiant.Nom, Etudiant.Prénom, Etudiant.Promo **From** Etudiant **Where** Etudiant.NumCarte = 'D001' ;

Nom	Prénom	Promo
Seck	Fatoumata	2

Cette requête donne le nom, le prénom et la promotion de l'étudiant qui a le numéro de carte D001.

IV. 2. 4. La jointure

Il est aussi possible d'afficher des informations venant de deux ou plusieurs tables différentes avec une seule requête, dans ce cas on parle de **jointure** entre tables.

Sa syntaxe est :

Select liste, attributs **From** liste, tables **Where** ConditionJointure [and ConditionSélection] ;

Exemple : Si en plus da la table **Etudiant**, nous avons la table **Classe** dont le contenu est :

Classe

Promo	NomFilière	Année
1	Agroforesterie	2008 - 2009
2	Info Appliquée	2009 - 2010
3	Agroforesterie	2010 - 2011

1. **Select** Nom, Prénom, Niveau, Année **From** Etudiant, Classe **Where** Etudiant.Promo = Classe.Promo **AND** Etudiant.Promo = 1 ;

Nom	Prénom	Niveau	Année
Diatta	Souléymane	L1	2008 - 2009
Gueye	Mamadou	L2	2008 - 2009

Cette requête donne le nom, le prénom, l'année académique et le niveau durant cette année des étudiants de la première promotion.

2. **Select** * **From** Etudiant, Classe **Where** Etudiant.Promo = Classe.Promo ;

NumCarte	Nom	Prénom	Niveau	Etudiant. Promo	NomFilière	Année	Classe.Promo
C001	Diatta	Souléymane	L1	1	Agroforesterie	2008 - 2009	1
B002	Gueye	Mamadou	L2	1	Agroforesterie	2008 - 2009	1
D001	Seck	Fatoumata	M1	2	Info Appliquée	2009 - 2010	2
E004	Traoré	Souléymane	L1	2	Info Appliquée	2009 - 2010	2

Cette requête donne l'ensemble des informations gardées dans les 2 tables sur chaque étudiant.

IV. 2. 5. Les fonctions d'agrégat et de mise en ordre

Il existe un certain nombre de fonctions qui permettent de :

- Faire des calculs sur les valeurs des attributs d'une table : **avg()** permet de calculer une moyenne, **sum()** permet de calculer une somme. Ces deux fonctions ne s'appliquent que sur des attributs de type **numérique** (entier ou réel) ;
- Compter le nombre de lignes qui répondent à un critère donné (**count()**) ;
- Avoir la plus petite (**min()**) ou la plus grande (**max()**) valeur d'une colonne ;
- D'ordonner ou de regrouper des enregistrements selon certaines conditions : **Group By** permet de regrouper des enregistrements suivant un ou plusieurs attributs, **Order**

By permet d'ordonner le résultat d'une requête suivant un ou des attributs (l'ordre peut être ascendant avec **Asc** ou descendant avec **Desc**).

Remarque : On peut ajouter l'option **Distinct** pour éliminer les doublons dans le résultat.

Having : La clause **Having** est toujours utilisée avec le groupage (**Group By**). Elle permet d'appliquer une restriction sur les groupes créés par la clause **Group By**.

Exists : Le prédicat **Exists** permet de vérifier l'existence ou non de données dans une sous-requête. Si cette dernière retourne au moins une ligne, le prédicat est vrai, il est faux sinon. Il peut être accompagné de la négation **Not**.

All : L'opérateur **All** permet de comparer la valeur d'un attribut avec un ensemble de valeurs. L'opérateur de comparaison ne peut cependant pas être l'égalité.

Any : L'opérateur **Any** permet de chercher une valeur dans une liste de valeurs.

Exemple : Supposons les tables suivantes :

Etudiant (Matricule, Nom, Prénom, Moy_Sem1, Moy_Sem2, #Classe)

Classe (Code, Nom, Nb_Etudiant)

1. **Select avg**(Moy_Sem1) **AS** Moy_Classe **From** Etudiant ;
2. **Select count**(Matricule) **AS** Nb_Moyenne **From** Etudiant **Where** Moy_Sem1 >= 10 ;
3. **Select min**(Moy_Sem2) **AS** Plus_Petite_Moyenne **From** Etudiant ;
4. **Select max**(Moy_Sem2) **AS** Plus_Grande_Moyenne **From** Etudiant ;
5. **Select Distinct** Etudiant.Nom, Prénom, Classe.Nom **From** Etudiant, Classe **Where** Etudiant.Classe = Classe.Code ;
6. **Select** Nom, Prénom **From** Etudiant **Where** Nom **Like** '?S%' **Order By** Nom **Asc**;
7. **Select** Nom, Nb_Etudiant **From** Classe **Group By** Nom **Having** Nb_Etudiant < 50 ;
8. **Select** * **From** Etudiant **AS** Etu_1 **Where Exists** (**Select** Nom, Prénom **From** Etudiant **AS** Etu_2 **Where** Etu_1.Nom = Etu_2.Nom **and** Etu_1.Prénom = Etu_2.Prénom **and** Etu_1.Classe = Etu_2.Classe) ;
9. **Select** * **From** Etudiant **Where** (Promo = 2) **and** (Age > **All** (**Select** Age **From** Etudiant **Where** Promo = 1) ;
10. **Select** * **From** Etudiant **Where** (Promo = 3) **and** (Age = **Any** (**Select** Age **From** Etudiant **Where** Promo = 1) ;

IV. 2. 6. Union, Intersection, Différence

Les opérations de l'algèbre relationnelle comme l'Union, l'Intersection et la différence peuvent être utilisées en SQL. Ces opérations ne sont pas normalisées mais la plupart des SGBD les implémentent. Elles correspondent respectivement aux opérateurs **Union**, **Intersect** et **Minus**. Leur syntaxe est :

Select …

Opérateur

Select …

Exemple :

Select Nom **From** Etudiant **Union** **Select** NomF **From** Filière ;	**Select** Prénom **From** Etudiant **Intersect** **Select** NomF **From** Filière ;

IV. 3. Mise à jour d'une base de données

Le rôle essentiel d'une base de données étant de permettre le stockage d'informations susceptible de changer très fréquemment durant son utilisation, il est indispensable de disposer d'outils permettant d'apporter ces changements sur les données de la base. La mise à jour d'une base comprend les opérations d'insertion, de suppression et de modification.

IV. 3. 1. Propriétés de la mise à jour

Contrairement à l'extraction d'informations, la mise à jour d'une base de données :

- Peut échouer alors que la syntaxe de la requête est correcte ;
- Echoue si elle porte sur 2 tables différentes en même temps ;
- Est totalement exécutée ou annulée, c'est donc une transaction ;
- Ne peut être annulée si l'on se trompe.

IV. 3. 2. L'insertion

L'insertion est l'opération qui permet d'enregistrer de nouveaux enregistrements dans la base. Sa syntaxe est :

Insert Into Nom_Table [(liste des colonnes)] **Values** (liste des valeurs) ;

Remarque :

- La liste des colonnes visées peut être omise à condition que l'ordre d'insertion concerne toutes les colonnes de la table.

- L'ordre des valeurs doit être le même que celui des colonnes visées même si la liste des colonnes est omise.
- Il est possible d'insérer plusieurs lignes en même temps.

Exemple :

Insert Into Classe **Values** ('CLINFO', 'Informatique', 150) ;

Insert Into Etudiant (Matricule, Nom, Prénom, Moy_Sem1, Moy_Sem2, Code_Classe) **Values** ('00FAF0034', 'Diouf', 'Malick', 12.5, 11.85, 'CLINFO') ;

IV. 3. 3. La modification

Les données enregistrées dans une base de données peuvent changer une ou plusieurs fois au cours de son utilisation. La modification permet de faire ces changements. Sa syntaxe est :

Update Nom_Table **SET** $Att_1 = V_1$, $Att_2 = V_2$, ….., $Att_3 = V_3$ [**Where** condition] ;

Remarque : La clause **Where** n'est pas obligatoire. Si elle est absente, tous les enregistrements de la table sont modifiés.

Exemple :

Update Etudiant **SET** Moy_Sem1 = 12.75 ;

Update Classe **SET** Nom = 'Informatique pure' **Where** Code = 'CLINFO' ;

IV. 3. 4. La suppression

Elle permet d'enlever de la base des informations qui ne sont plus utiles ou qui sont erronées. Sa syntaxe est :

Delete From Nom_Table [**Where** condition] ;

Remarque : Si la condition **Where** n'est pas spécifiée, tous les enregistrements de la table sont supprimés.

Exemple :

Delete From Etudiant **Where** Matricule = '00FAF0034' ;

V. Langage de Contrôle de Données (DCL) :

V. 1. Création et suppression d'utilisateurs

Plusieurs personnes peuvent travailler simultanément sur une même base de données. Cependant chacun doit avoir :

- un nom d'utilisateur et un mot de passe ;
- des opérations spécifiques qu'il peut effectuer (ses privilèges ou droits) ;
- des informations qu'il peut visualiser ou mettre à jour.

Pour cela il faut d'abord créer un compte d'utilisateur pour chaque personne devant accéder à la base. La syntaxe de la création d'un utilisateur est :

Create user login **Identified By** 'mot_de_passe' ;

Exemple :

1. **Create user** user1 **Identified By** '01234' ;
2. **Create user** user2 **Identified By** '56789' ;
3. **Create user** user3 **Identified By** 'abcde' ;

La suppression d'un utilisateur est faite en utilisant la syntaxe suivante :

Drop User login ;

Exemple :

Drop User user3 ;

V. 2. Modification de logins ou de mots de passe d'utilisateurs

La modification du login ou du mot de passe d'un utilisateur se fait avec l'ordre **Alter User**. La syntaxe de la modification est :

Alter User NomUser **With** NouveauLogin ;

Alter User NomUser **Identified By** NouveauPass ;

Exemple :

Alter User user1 **With** user5 ;

Alter User user5 **Identified By** 'fghij' ;

V. 3. Attribution de privilèges

Les droits que l'ont peut donner aux utilisateurs sont les suivants ;

- **SELECT :** droit de visualiser le contenu ;
- **INSERT :** droit d'effectuer des insertions ;
- **UPDATE :** droit d'effectuer des mises à jour ;
- **DELETE :** droit d'effectuer des suppressions d'enregistrements dans des tables ;

- **CREATE :** droit de créer des bases ou des tables ;
- **DROP :** droit de supprimer des bases ou des tables ;
- **REFERENCES :** droit lié aux clés étrangères "foreign keys" ;
- **INDEX :** droit de créer ou détruire des index de tables ;
- **ALTER :** droit de modifier la structure des tables ;
- **CREATE VIEW :** droit de créer des vues ;
- **SHOW VIEW :** droit de visualiser les vues créées ;
- **TRIGGER :** droit de créer des triggers ;
- **USAGE :** droit de se connecter au serveur, sans rien faire d'autre (utile uniquement pour changer le mot de passe de connexion) ;
- **LOCK :** droit de verrouiller/déverrouiller des tables ;
- **GRANT :** permet d'affecter des droits et permission à un utilisateur ;
- **REVOKE :** permet de retirer des droits à un utilisateur ;
- **Privilèges globaux** : C'est des privilèges qui ne s'appliquent pas à UNE base particulière :
- **RELOAD :** permission de relancer le serveur MySQL et d'écrire les tables sur disques.
- **SHUTDOWN :** droit d'arrêter le serveur "mysqld" ;
- **PROCESS :** droit de contrôler les processus utilisateurs ;
- **FILE :** droit d'écrire ou lire dans des fichiers ASCII avec les commandes "**load data**" et "**into outfile**".

La syntaxe de l'attribution de droits à un utilisateur est :

Grant liste_de_privilèges **ON** liste_de_tables **TO** utilisateur ;

Exemple :

4. **Grant** Select **ON** Etudiant **TO** user1 ;
5. **Grant** Delete, Update, Insert, Select **ON** Etudiant, Classe **TO** user2 ;
6. **Grant** All Privileges **On** NomBase.* **TO** user3.

V. 4. Suppression de privilèges

La suppression de privilèges se fait avec l'ordre **REVOKE**. Sa syntaxe est :

Revoke liste_de_privilèges **ON** liste_de_tables **FROM** utilisateur ;

Exemple :

Revoke Delete, Insert **On** Etudiant **From** user2 ;

Remarque : Les droits sont cumulatifs et un même droit reçu plusieurs fois ne peut être