Chapitre : Le Langage SQL

I. Introduction

SQL (**Structured Query Language**, ou **langage de requêtes structurées**). C'est un langage déclaratif dans le sens où il suffit d'exprimer ce que l'on veut obtenir, sans devoir expliquer la méthode pour y parvenir. Ainsi, les opérations directement utilisables par les usagers sont en général celles des langages dits assertionnels.

Ces langages, bâtis sur l'algèbre relationnelle, permettent de manipuler des bases de données relationnelles. Nous pouvons citer à titre d'exemple :
- QUEL (QUEry Language) par Zook en 1977
- QBE (Query By Example) par Zloof en 1977
- SQL (Structured Query Language) par IBM

Aujourd'hui, le langage SQL est normalisé et constitue le standard d'accès aux bases de données relationnelles.

II. Présentation de SQL

SQL est un langage de gestion de bases de données relationnelles. Il étend l'algèbre relationnelle et permet :
- de créer des bases de données
- d'ajouter, modifier et consulter des données d'une base de données existante
- de contrôler les accès aux informations d'une BD

SQL est un langage ensembliste qui :
- respecte l'indépendance des niveaux
- garantit la sécurité
- permet une gestion multi-utilisateurs
- respecte les contraintes d'intégrité

SQL étend l'algèbre relationnelle, cependant il utilise les termes **table**, **ligne** et **colonne** au lieu des termes relationnels **relation**, **tuple** et **attribut**. Une instruction SQL peut s'écrire sur plusieurs lignes. Pour être exécutée, l'instruction doit se terminer par un point-virgule. SQL est un langage de requête, pas un langage de programmation. Par conséquent, il n'y pas de variables ni de structures de contrôles (if, while, for). C'est pourquoi on peut éventuellement l'intégrer dans un langage de programmation comme PL/SQL, C ou Java.

III. Les 3 facettes de SQL

SQL se subdivise en 3 sous-langages :
- **LDD** (*Langage de Définition de Données*) : création, modification et suppression des objets que peut manipuler une BD (tables, vues et index, etc).
- **LMD** (*Langage de Manipulation de Données*) : ajout, suppression, modification et extraction des données.
- **LCD** (*Langage de Contrôle de Données*) : sécurisation et validation des données.

Chacun de ces sous-langages propose ses mots-clés propres. Voici les principales primitives que nous verrons au cours de ce chapitre :

LDD	LMD	LCD
CREATE	SELECT	GRANT
ALTER	INSERT	REVOKE
DROP	UPDATE	COMMIT
	DELETE	ROLLBACK

IV. Le Langage de Définition de Données (LDD)

Définition :
Le **langage de définition de données** permet la définition du schéma d'une base de données ainsi que certaines de ses **contraintes d'intégrité**. Il comporte également les opérations de mise à jour de schéma telles que la suppression d'une table ou la modification de la définition d'une table.

Le LDD est donc composé de 3 primitives :
- **CREATE** pour la création d'un objet (table, index, vue, etc).
- **ALTER** pour la modification d'un objet.
- **DROP** pour la suppression d'un objet.

Dans ce chapitre on s'intéresse au langage de manipulation de Données

V. Le Langage de Manipulation de Données (LMD)

Définition :

Le **Langage de Manipulation de Données** permet de :

- afficher ;
- insérer ;
- mettre à jour ;
- supprimer des données dans les tables.

Le LMD est donc composé de 4 primitives :

- **SELECT** pour l'affichage des données des tables.
- **INSERT** pour l'insertion de données dans les tables.
- **UPDATE** pour la mise à jour des données.
- **DELETE** pour la suppression des données.

1. *SELECT*

L'affichage des données d'une ou plusieurs tables se fait grâce au mot-clé **SELECT**.

Syntaxe de la primitive SELECT en SQL

```
SELECT * | [DISTINCT | ALL] <attributs>
FROM <tables>
  [WHERE <condition logique de niveau tuple>]
  [GROUP BY <attributs>
  [HAVING <condition logique de niveau groupe>]]
  [ORDER BY <attributs>];
```

Notation (les éléments entre [ ] sont optionnels) :

- Le symbole "|" signifie "ou" ; ainsi on peut faire suivre le mot-clé **SELECT** par une * OU par <attributs> précédé le cas échéant par **DISTINCT** ou **ALL**.

La clause **SELECT** décrit la relation résultat, <attributs> désigne :

- soit une liste d'attributs ;
- soit une expression obtenue à l'aide des fonctions d'agrégation **SUM** (somme), **AVG** (moyenne), **COUNT** (compte), **MIN** et **MAX** ;
- soit une expression.

Les mots-clés **DISTINCT** et **ALL** permettent respectivement d'éliminer ou de conserver les doublons après une projection mise en œuvre par l'ordre **SELECT** suivi d'une liste d'attributs.

La clause **FROM** désigne les relations concernées par la requête. Chaque nom de relation peut être suivi d'un alias (variables synonymes pour chaque relation). Cet alias permet parfois des économies d'écritures, mais il permet surtout de lever des ambiguïtés sur les noms d'attributs (plusieurs relations qui possèdent le même nom d'attribut). Des exemples de définition d'alias sont proposés dans la partie présentant des exercices corrigés.

La clause **WHERE**, optionnelle, spécifie les critères de sélection. La condition logique de niveau tuple est une expression logique spécifiant les prédicats de restriction ou de jointure à satisfaire par la réponse. On peut également insérer dans la clause **WHERE**, une nouvelle instruction **SELECT** (requête imbriquée) ce qui permet de travailler sur un ensemble de tuples restreints par cette sélection.

Lorsque ces critères portent sur des fonctions de groupe, on utilise la clause **HAVING**.

La clause **GROUP BY** partitionne la relation en groupes. La liste des attributs spécifiés dans cette clause indique le critère de groupement.

La clause **ORDER BY** permet de trier les résultats obtenus selon un tri ascendant (**ASC**) ou descendant (**DESC**). On veillera à choisir des colonnes de tri présent dans les attributs du **SELECT**.

a. La clause SELECT et la clause FROM

La clause **SELECT** et la clause **FROM** sont liées. En effet, lors de chaque opération de sélection, le mot-clé **FROM** permet d'identifier la ou les tables nécessaires à la construction du résultat de la requête.

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) ; F pour Fournisseur
P (NP, nomP, Poids,Couleur) ; P pour Produit
U (NU, nomU, ville) ; U pour Usine
PUF (NP, NU, NF, qt) ; PUF pour Livraison

1- Liste de tous les fournisseurs……

SELECT * FROM F ;

Remarque : le symbole * dans la clause **SELECT** signifie qu'on désire récupérer TOUS les champs de la table.

SELECT Ville FROM F ;

Cette instruction SQL permet d'afficher le contenu du champ Ville des tuples de la table F.

Remarque : la requête ne porte que sur un seul champ, cela équivaut à une projection de la table *F* sur ce champ. On peut ajouter autant d'attributs que l'on désire afficher.

1- 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.

SELECT DISTINCT Ville FROM F;

b. Les prédicats

Une condition est appelée **prédicat** en SQL. Un prédicat permet de comparer 2 expressions de valeurs :

- la première expression contenant des spécifications de colonnes est appelée **terme** ;
- la seconde expression contenant seulement des spécifications de constantes est appelée **constante**.

Il existe une grande diversité de prédicats en SQL, on trouve en effet :

1. un prédicat de comparaison permettant de comparer un terme à une constante à l'aide des opérateurs suivants :
   - = : égal
   - <> ou != : différent
   - > : plus grand que
   - >= : plus grand ou égal
   - < : plus petit que
   - <= : plus petit ou égal

   Ces opérateurs sont valables pour les types **numeriques**.
2. un prédicat d'intervalle **BETWEEN** permettant de tester si la valeur d'un terme est comprise entre la valeur de 2 constantes ;
3. un prédicat de comparaison de texte noté **LIKE** permettant de tester si un terme de type chaîne de caractères contient une ou plusieurs sous-chaînes (ce prédicat est donc réservé aux types chaine de caractère). Le caractère % remplace une chaîne de caractères quelconque y compris la chaîne vide, le caractère _ remplace n'importe quel caractère unique ;
4. un prédicat de test de nullité qui permet de tester si un terme à une valeur convenue **NULL**, signifiant que sa valeur est inconnue ou que le champ n'a pas été renseigné. Pour tester la nullité, on utilise **IS NULL**, et la non nullité par **IS NOT NULL** ;
5. un prédicat d'appartenance noté **IN** qui permet de tester si la valeur d'un terme appartient à une liste de valeurs constantes.

c. La clause WHERE

La clause **WHERE** spécifie une condition de sélection. Une condition de sélection définit un critère qui, appliqué à un tuple, est vrai, faux ou inconnu (mettant en cause un attribut de valeur **NULL**). Cette condition peut inclure des opérateurs booléens (**AND**, **OR**, **NOT**), des conditions élémentaires et des parenthèses.

Les tableaux ci-après donnent les tables de vérité permettant de calculer les valeurs de vérité d'une condition de sélection. Seuls les tuples satisfaisant la condition de sélection sont pris en compte par la requête.

AND	Vrai	Faux	Inconnu
Vrai	Vrai	Faux	Inconnu
Faux	Faux	Faux	Faux
Inconnu	Inconnu	Faux	Inconnu

OR	Vrai	Faux	Inconnu
Vrai	Vrai	Vrai	Vrai
Faux	Vrai	Faux	Faux
Inconnu	Vrai	Faux	Inconnu

NOT	Vrai	Faux	Inconnu
	Faux	Vrai	Inconnu

Prédicats divers :

- SELECT * FROM P
WHERE Prix BETWEEN 100 AND 300 ;

Cette instruction SQL permet d'afficher les tuples de la table PRODUIT Dont le Prix est compris entre 100 et 300.

- SELECT * FROM P
WHERE NomP LIKE 'S%'
AND Couleur IN ('Bleu', 'Rouge', 'Vert')
AND Poids IS NOT NULL;

Cette instruction SQL permet d'afficher tous les tuples de la table PRODUIT dont le Nom commence par 'S', dont la couleur est soit `Bleu, Rouge ou Vert` et dont le Poids n'est pas inconnu (donc connu) :

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 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
               FROM       PUF                       NP
                                                 livrés par ce
               WHERE      PUF.N    = F.NF)     fournisseur */
                            F
```

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

Remarque : la renommage crée une copie de la relation PUF nommée PUFX.

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
            n
om 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- Les fonctions d'agrégation :

Les fonctions d'Agrégation (MAX, MIN, SUM, AVG, COUNT) peuvent être utilisées dans les clauses SELECT, WHERE et HAVING. Les fonctions d'agrégation sont appliquées à l'ensemble d'une colonne (ou d'un groupe) et fournissent une valeur unique.

7.1- La fonction MAX et la fonction MIN

Les fonctions MAX et MIN renvoient respectivement le maximum et le minimum d'un champ.

7-2 La fonction COUNT

La fonction **COUNT** comptabilise le nombre de lignes pour lesquelles l'expression est non **NULL**. Une * est souvent utilisée avec **COUNT** pour indiquer le nombre de lignes. Si on ne veut pas compter plusieurs fois les valeurs identiques d'une même colonne, il faut utiliser le mot-clé **DISTINCT** suivi du nom de la colonne (sauf s'il s'agit d'un attribut clé de la relation).

7-3 La fonction SUM

La fonction **SUM** effectue, pour un ensemble de tuples, la somme des valeurs d'un attribut. Cette fonction est uniquement utilisable pour le type **NUMBER**.

7-4 La fonction AVG

La fonction **AVG** calcule, pour un ensemble de tuples, la moyenne arithmétique des valeurs d'un attribut. Cette fonction est uniquement utilisable pour le type **NUMBER**. Elle vérifie la formule suivante : **AVG** = Somme des valeurs non **NULL** / nombre de valeurs non **NULL**.

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

```
SELECT    COUNT (*)          /* on compte les tuples de PUF
FROM        PUF                  tels 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
```

Exemple : Le prix total des produits rouge.

```
SELECT SUM (Prix)
FROM P
WHERE Couleur = 'Rouge' ;
```

8- *Recherche avec partition des tuples d'une relation*

8-1 La clause GROUP BY

La clause **GROUP BY** permet de partitionner la relation résultat selon les valeurs d'un ou de plusieurs attributs. Les seuls noms de colonnes (en dehors des fonctions statistiques) qui peuvent apparaître dans le **SELECT** sont celles qui figurent dans le **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 :

i. Classer les tuples de PUF, groupe par groupe (un groupe = ensemble des tuples ayant même NF),

ii. Pour chaque groupe :
- évaluer le résultat du SELECT (compter le nombre de NP différents dans ce groupe).

8-2 La clause HAVING

La clause **HAVING** définit les conditions que les groupes doivent respecter pour être retenus, elle sélectionne les partitions désirées. Elle ne peut comprendre que des conditions dont le premier terme est une fonction statistique. La clause **HAVING** est aux groupes (**GROUP BY**) ce que la clause **WHERE** est aux lignes (**SELECT**).

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) ,
et{Ak1, Ak2, … , Akw} $\supseteq$ {Ai1, Ai2, … , Aiu} ,
et {Ak1, Ak2, … , Akw} $\cap$ {Aj1, Aj2, … , Ajv} = Ø .

9- Un Résultat trié :

La clause ORDER BY

La clause **ORDER BY** permet de trier les tuples du résultat final. La clause **ORDER BY** permet d'ordonner la relation résultat sur un ou plusieurs attributs, l'ordre pouvant être croissant (grâce au mot-clé **ASC**) ou décroissant (grâce au mot-clé **DESC**), **ASC** étant l'ordre par défaut.

Remarque : Dans une requête SQL, la clause **ORDER BY** se situe juste après une clause **WHERE** ou après une clause **GROUP BY**.

Exemple : liste des fournisseurs de JIJEL par ordre alphabétique.

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

La clause **ORDER BY** permet également un tri sur plusieurs attributs : le tri sera effectué prioritairement sur le premier attribut, puis sur le second et ainsi de suite.

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

10- Les opérateurs relationnels

Les opérateurs relationnels en SQL sont basés sur les opérateurs relationnels vus dans le chapitre Modèle Relationnel. En SQL, les principaux opérateurs sont UNION, INTERSECTION, MINUS (différence). Les deux relations R1 et R2 sur lesquelles sont appliqués les opérateurs UNION, INTERSECTION et MINUS doivent absolument avoir avoir le même schéma de table (nombre équivalent d'attributs et attributs identiques deux à deux).

11_ Les alias

Les alias concourent à améliorer la lisibilité et la concision d'une requête. Il existe deux types d'alias : les alias de tables et les alias de champs. Ils peuvent également s'appliquer à une fonction d'agrégation retournant des données sous forme de colonnes.

```
SELECT Alias_table.nom_champ AS Alias_champ
   FROM nom_table AS Alias_table;
```

La clause AS, affectant un alias à une table ou une colonne, peut être remplacée par un simple espace blanc (Pour le renommage)

Les alias de champs peuvent être des chaînes de caractères composées de mots et d'espaces. Dans ce cas, il faut placer l'alias entre simple guillemet.

```
SELECT Alias_table.nom_champ AS 'Un alias de champ'
FROM nom_table AS Alias_table;
```

Lors de l'affichage des lignes résultantes, **les alias de champs se substitueront aux noms de colonnes**. De cette manière, il devient possible d'attribuer des noms de colonnes plus explicites pour les utilisateurs de la base de données.

Les alias sont particulièrement **utiles dans le cadre des jointures et des requêtes imbriquées**. Dans les deux cas, il devient possible de faire appel à des champs de noms identiques en les distinguant par des préfixes qui sont les alias de table.

11- Mise à jour de la base de données

11-1 Insérer des tuples

- Insérer un tuple :
  INSERT INTO nomrelation VALUES < 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.
```

11-2 Supprimer des tuples

DELETE nom_relation [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.

11-3 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"
```

Soit la base de données relationnelle, PUF, de schéma :

U (NU, NomU, Ville)
P (NP, NomP, Couleur, Poids)
F (NF, NomF, Statut, Ville)
PUF (NP, NU, NF, Quantité)

décrivant les faits suivants :

- **U:** une usine est décrite par son numéro NU, son nom NomU, la ville Ville dans laquelle elle est située;
- **P:** un produit est décrit par son numéro NP, son nom NomP, sa Couleur, son Poids;
- **F:** un fournisseur est décrit par son numéro NF, son nom NomF, son Statut (fournisseur sous-traitant, fournisseur-exclusif,.....), la Ville où il est domicilié;
- **PUF:** le produit de numéro NP a été livré à l'usine de numéro NU par le fournisseur de numéro NF dans une Quantité donnée.

Exprimer en SQL et en AR (Si possible) les requêtes suivantes:

1) Donner le numéro, le nom et la ville de toutes les usines.

EN SQL

.

```
SELECT NU, NOMU, Ville
 FROM U ;
```

Ou bien :

```
SELECT *  FROM U ;
```

EN AR :

RES := $\pi_{NU, NOMU, Ville}(U)$;

2) Donner le numéro, le nom et la ville de toutes les usines d'Alger.
EN SQL

```
SELECT NU, NOMU, Ville
FROM U
WHERE Ville = 'ALGER' ;
```

EN AR :

RES := $\pi_{NU, NOMU, Ville}(\sigma_{Ville = 'ALGER'}(U))$;

3) Donner les numéros des fournisseurs qui approvisionnent l'usine n°1 en produit n°1.
4) Donner le nom et la couleur des produits livrés par le fournisseur n°1.
5) Donner les numéros des fournisseurs qui approvisionnent l'usine n°1 en un produit rouge.
6) Donner les noms des fournisseurs qui approvisionnent une usine d'Alger ou de Jijel en un produit rouge.
7) Donner les numéros des produits livrés à une usine par un fournisseur de la même ville.
8) Donner les numéros des produits livrés à une usine de Jijel par un fournisseur de Jijel.
9) Donner les numéros des usines qui ont au moins un fournisseur qui n'est pas de la même ville.
10) Donner les numéros des fournisseurs qui approvisionnent à la fois les usines n°1 et n°2.
11) Donner les numéros des usines qui utilisent au moins un produit disponible chez le fournisseur n°3 (c'est- à-dire un produit qu'il livre mais pas nécessairement à cette usine).
12) Donner le numéro du produit le plus léger (les numéros si plusieurs produits ont ce même poids).
13) Donner les numéros des usines qui ne reçoivent aucun produit rouge d'un fournisseur Jijilien.
14) Donner les numéros des fournisseurs qui fournissent au moins un produit fourni par au moins un fournisseur qui fournit au moins un produit rouge.

---

15) Ajouter un nouveau fournisseur : <45, Ahmed, sous-traitant, Alger>
16) Supprimer tous les produits de couleur noire et de numéro compris entre 100 et 199.
17) Changer la ville du fournisseur n°1 : il a déménagé à Constantine.