Les Langages de manipulation relationnels

1. Pourquoi des langages de manipulation de données relationnelles ?

La structure des relations étant semblable à celle des tableaux en mémoire centrale, on peut se demander pourquoi avoir inventé des langages spéciaux d'interrogation et de mise à jour pour les relations. Ne pouvait-on pas employer n'importe quel langage de programmation ? Cela aurait l'avantage de ne pas obliger les utilisateurs à apprendre un autre langage spécifique des bases de données.

L'inconvénient dans une telle solution est que les programmes des utilisateurs doivent connaître l'organisation des tuples dans les relations pour pouvoir accéder plus rapidement aux informations dont ils ont besoin. L'indépendance recherchée dans les SGBDs, des programmes et des données n 'est plus réalisée. Les utilisateurs doivent connaître l'organisation des données (existence de tri, d'index,…), et il est impossible de changer cette organisation (à des fins d'optimisation par exemple) sans affecter les programmes des utilisateurs.

Les langages de manipulation de données (LMD) doivent donc être purement déclaratifs, c'est à dire ne porter que sur les concepts du schéma conceptuel (relations, attributs, domaines) et ignorer tout de l'organisation interne des relations. Cependant, ils doivent aussi être efficaces, c'est à dire avoir des temps de réponse courts même si la base de données est très grande. C'est pourquoi, les LMD offrent un éventail de fonctions limité à celles qu'on sait optimiser, mais assez vaste pour permettre d'exprimer la plupart des requêtes.

2. LMD algébriques et LMD prédicatifs

Le modèle relationnel a été à l'origine proposé avec deux LMD de base, l'algèbre relationnelle et le calcul des tuples, équivalents en puissance et qui ont fixé l'ensemble des fonctions de base que tout LMD relationnel doit offrir. A partir de ces deux langages, d'autres LMD ont pu être définis, qui sont plus conviviaux pour les utilisateurs, et qui ont au moins la même puissance que l'algèbre ou le calcul.

En plus des fonctions de l'algèbre ou du calcul, ces LMD offrent généralement des possibilités de mise à jour de la base de données, et d'emploi, dans les requêtes, d'expressions arithmétiques et de fonctions d'agrégation telles que la cardinalité, somme, minimum, maximum et moyenne.

On dit qu'un LMD est complet s'il offre au moins les mêmes fonctions que l'algèbre ou le calcul des tuples.

L'intérêt de l'algèbre relationnelle, outre le fait qu'elle a donné naissance à des LMD de type algébrique, est d'avoir identifié les opérateurs fondamentaux d'utilisation d'une base de données relationnelles. Ces opérateurs définissent aussi les principales fonctions à optimiser dans les SGBD relationnels.

Un autre type de LMD est constitué des langages issus du calcul des prédicats de la logique du premier ordre. L'objectif des langages prédicatifs est d'exprimer une requête par la simple définition du résultat, an faisant abstraction du mécanisme utilisé par le SGBD pour la construction de ce résultat. Ce sont donc des langages déclaratifs. Deux adaptations du calcul des prédicats au modèle relationnel ont été proposées, selon que les variables utilisées dans les formules du calcul désignent des tuples d'une relation ou des valeurs dans un domaine ; toutes deux ont conduit à des langages utilisateurs :

- Le calcul des tuples qui a donné naissance au LMD QUEL du SGBD relationnel INGRES
- Le calcul des domaines qui a donné naissance au LMD de type graphique, QBE, proposé par IBM.

SQL qui est le LMD relationnel le plus répandu du fait que c'est la seule norme existante pour les LMD relationnels, comporte des caractéristiques de type algébrique et d'autres de type prédicatif.

Dans ce cours, nous étudions successivement l'algèbre relationnelle et le langage SQL.

Partie 1 : L'algèbre relationnelle

1. Introduction

L'algèbre relationnelle est un ensemble d'opérateurs qui, à partir d'une ou deux relations existantes, créent en résultat une nouvelle relation temporaire (c'est à dire qui a une durée de vie limitée , généralement, la relation est détruite à la fin du programme utilisateur ou de la transaction qui l'a créée). La relation résultat a exactement les mêmes caractéristiques qu'une relation de la base de données et peut donc être manipulée à nouveau par les opérateurs de l'algèbre.

Formellement, l'algèbre comprend :

- Cinq opérateurs de base : sélection, projection, union, différence et produit.
- Un opérateur syntaxique, renommer, qui ne fait que modifier le schéma et pas les tuples.

A partir de ces opérateurs, d'autre opérateurs ont été proposés, qui sont équivalents à la composition de plusieurs opérateur de base. Ces opérateurs déduits, sont des raccourcis d'écriture, qui n'apportent aucune fonctionnalité nouvelle, mais qui sont pratiques lors de l'écriture des requêtes. Les opérateurs déduits les plus fréquents sont : intersection, jointure naturelle, thêta-jointure et division.

Les opérateurs de l'algèbre peuvent être regroupés en deux classes :

- Les opérateurs provenant de la théorie mathématique sur les ensembles (applicables car chaque relation est définie comme un ensemble de tuples) : union, intersection, différence, produit.
- Les opérateurs définis spécialement pour les bases de données relationnelles : sélection, projection, jointure, division et renommage.

2. Les opérateurs :

2.1 Projection :

Cet opérateur construit une relation résultat où n'apparaissent certains attribut de la relation opérande (en termes de tableau, cela revient à extraire certaines colonnes).

Définition :

Soit R(A1, A2,……..,An) une relation, et soit Ai1, Ai2,…., Aij un sous-ensemble de ses attributs. La projection de R sur Ai1, Ai2, A…., Aij, notée : Π[Ai1, Ai2, …., Aij] R, crée une nouvelle relation, temporaire de schéma (Ai1, Ai2, A…., Aij) et de population égale à l'ensemble des tuples de R tronqués à Ai1, Ai2, A…., Aij.

Remarque :

Le résultat est un ensemble de tuples, c'est à dire que si la projection crée des tuples en double, (cas d'une projection éliminant tous les identifiants de R), ces doubles seront supprimés automatiquement.

Exemple : soit la relation

Personne

Nom	Prénom	Jour-Nais	Mois-Nais	An-Nais	Sexe
Mecheri	Mohammed	30	07	75	M
zouaghi	Ouarda	20	11	75	F
Bella	Ahlem	13	05	76	F
Zouaghi	Anouar	27	03	75	M
Achouri	Ahmed	10	10	77	M
Ouali	Rachid	05	03	74	M

On construit l'ensemble des noms et prénoms des personnes avec l'opération :

NP := Π [nom, prénom] Personne

On obtient :

NP

Nom	Prénom
Mecheri	Mohammed
zouaghi	Ouarda
Bella	Ahlem
Zouaghi	Anouar
Achouri	Ahmed
Ouali	Rachid

L'opération **NA := Π[nom, an-nais] Personne** donnera en résultat :

NA

Nom	An-Nais
Mecheri	75
Bella	76
Zouaghi	75
Achouri	77
Ouali	74

2.2 Sélection :

Cet opérateur construit une relation résultat où n'apparaissent que certains tuples de la relation opérande (en termes de tableau, cela revient à extraire certaines lignes). Les tuples retenus sont ceux satisfaisant une condition explicite, appelée prédicat de selection.

Définition :

Soit R(A1, A2,……..,An) une relation, la sélection de R selon un prédicat p , notée : σ [p] R, crée une nouvelle relation , temporaire, de schéma identique à celui de R, et de population l'ensemble des tuples de R pour lesquels le prédicat p est vrai.

Exemple : pour créer une relation Femmes, contenant l'ensemble des personnes de sexe féminin, on écrira : **Femmes := σ [sexe = 'F'] Personne**

Ce qui donne en résultat :

Femmes

Nom	Prénom	Jour-Nais	Mois-Nais	An-Nais	Sexe
zouaghi	Ouarda	20	11	75	F

Bella	Ahlem	13	05	76	F

Le prédicat de sélection permet de comparer la valeur d'attributs de R à celle d'autres attributs de R ou à des constantes. Sa forme est la suivante :

<p> ::= <condition> | <p> <opérateur logique> <p> | ¬<p> | ″(″<P>″)″

<opérateur logique> ::= ∧ |∨

<condition> ::= nom-attribut <opérateur comparaison> valeur |

nom-attribut < opérateur comparaison> nom-attribut

<opérateur comparaison> ::= = | ≥ | ≤ | ≠ | < | >

Les opérateurs de comparaison ≥, ≤, <, > ne peuvent être appliqués, qu'aux attributs dont les domaines contiennent des valeurs ordonnées (valeurs numériques, dates, chaînes de caractères). Les domaines de chaînes de caractères alphabétiques sont triés alphabétiquement, tandis que les domaines de chaînes de caractères alphanumériques sont triés selon les codes numériques des caractères. Si le domaine d'un attribut est un ensemble de valeurs non triées, les seuls opérateurs de comparaison utilisables sont = et |≠.

2.3 Expressions d'algèbre :

Les opérateurs de l'algèbre relationnelle peuvent être combinés dans des expressions pour exprimer des requêtes non élémentaires.

Exemple : on obtient la liste des noms et prénoms des hommes nés avant 1976 par l'expression :

Résultat := Π[nom, prénom] σ [sexe = 'M' ∧ an-nais < 1976] Personne

Résultat

Nom	Prénom
Mecheri	Mohammed
zouaghi	Ouarda
Zouaghi	Anouar
Ouali	Rachid

Le même résultat pourrait être obtenu en écrivant deux opérations l'une après l'autre en créant une relation intermédiaire (dans ce cas, il faut nommer les relation intermédiaires) :

Res-intermédiaire := σ [sexe = 'M' ∧ an-nais < 1980] Personne

Résultat := Π[nom, prénom] Res-intermédiaire

2.4Jointure (naturelle) de deux relations ayant au moins un attribut commun :

Définition :

Etant donné deux relations R(X, Y) et S(Y, Z), où X, Y, Z symbolisent soit un attribut, soit un ensemble d'attributs, et où Y n'est pas vide, la jointure naturelle de R et S notée

R * S, crée une nouvelle relation temporaire, de schéma (X, Y, Z). La population de R*S est l'ensemble des tuples <x, y, z> créés par composition d'un tuple <x, y> de R et d'un tuple <y,z> de s tels que les deux tuples ont la même valeur pour Y

Remarque : il existe plusieurs symboles pour l'opérateur de jointure naturelle, (* et).

On remarque que la population de R*S comporte N tuples, N ∈ [0..card(R) * card(S)] :

N= 0 sil n'existe pas de tuple de R et de S qui ont même valeur pour Y

N = card(R) * card(S), si les tuples de R et de S ont tous la même valeur y0 pour Y

Exemple :

Schéma relationnel « formation-permanente » de la base de données relationnelle suivante :
Relations :

- **Relation Personne (n°P, nom, adr)** : désigne tout étudiant et tout enseignant de l'institut.
- **Relation PersonnePrénoms (n°P, prénom)** ; identifiant externe : n°P référence une personne
- **Relation Etudiant (n°P, n°E, dateN)** : tout individu qui est actuellement inscrit à l'institut ou qui a déjà passé avec succès un des cours de l'institut. Clé candidate : n°P. clé externe : n°P référence une personne
- **Relation EtudiantEtudes (n°E, diplôme année)** : études antérieures des étudiants. Identifiant externe : n°E référence un étudiant n°E
- **Relation Enseignant (n°P, tel, statut, banque, agence, compte)** : tout individu assurant actuellement un ou plusieurs cours à l'institut. Identifiant externe :n°P référence une personne
- **Relation Cours (nomC, cycle, n°Ens)** : désigne tout cours offert par l'institut. Identifiant externe : n°Ens référence un enseignant
- **Relation Obtenu** (n°E, nomC, note, année). L'étudiant n°E a réussi le cours nomC telle année et a obtenu telle note. Identifiants externes : n°E référence un étudiant n°E et nomC référence un cours.
- **Relation inscrit (n°E, nomC).** Actuellement, l'étudiant n°E est inscrit au cours nomC. Identifiants externes : n°E référence un étudiant n°E et nomC référence un cours.

Exemple 1 : On désire tous les renseignements sur les étudiants (nom, adresse, date de naissance, numéro d'étudiant, et de personne :

*Réponse : Etudiant * Personne*

Exemple 2 : noms des étudiants ayant réussi le cours d'algorithmique :

*Réponse : Π [nom] (personne * Etudiant * σ [nomC = ''algo''] obtenu)*

2.4Renommer un ou des attributs d'une relation

L'opérateur renommer, noté α, permet de changer le nom ou de plusieurs attributs d'une relation R :

α [nom-attr1 :nouveau-nom-pour-attr1,……..] R

Cet opérateur est utile avant les jointures s'il y a un problème d'homonymie ou de synonymie, ou avant les opérateurs ensemblistes (union, différence, intersection) qui requièrent que les attributs correspondant aient le même nom.

2.5 La thêta jointure de deux relations selon la valeur d'un prédicat :

Définition : soient deux relations R(A1, A2, …., An) et T(B1, B2,…….Bp) n'ayant pas d'attribut de même nom, la thêta-jointure de R et T selon le prédicat p, notée : **R *[p] T** , crée une nouvelle relation temporaire de schéma (A1, A2, …., An, B1, B2,…….Bp), et de population égale à l'ensemble des tuples de R et de T concaténés qui satisfont le prédicat.

Le prédicat est de la même forme que le prédicat d'une sélection, sauf pour les conditions élémentaires qui comparent un attribut de r à un attribut de T :

<condition> ::=nom-attribut-R <opérateur de comparaison> nom-attribut-T

exemple :

liste des couples de numéros d'étudiants, tels que ces deux étudiats soient nés le même jour.

/* on crée d'abord une autre relation Etudiant avec des attributs renommés*/

réponse : Etudiant2 := α[n°E : n°E2, dateN : dateN2]$\prod$[n°E, dateN] Etudiant

$\prod$[n°E, n°E2] (Etudiant [n°E<n°E2 dateN= dateN2] Etudiant2)*

2.6 Union, Différence, Intersection de deux relations de même schéma

Définition : soient R et S deux relations de même schéma R(A1, A2,…., An) et S(A1, A2,..., An) :

Union : R$\cup$S crée une relation temporaire de même schéma et de population égale à l'ensemble des tuples de R et ceux de S (avec élimination des doubles éventuellement créés).

Différence : R-S crée une relation temporaire de même schéma et de population égale à l'ensemble des tuples de R moins ceux de S, c'est à dire les tuples qui se trouvent dans R mais pas dans S.

Intersection : R$\cap$S crée une relation temporaire de même schéma et de population égale à l'ensemble des tuples de R qui ont un tuple de même valeur dans S

Exemples :

1) liste des numéros de personnes qui sont soit enseignant de BD soit étudiant en BD

réponse :on crée deux relations temporaires EnsBD et EtudBD

EnsBD := α [n°Ens : n°P] $\prod$ [n°Ens] σ [nomC = ' BD'] cours

EtudBD := $\prod$ [n°P]etudiant σ [nomC = ' BD'] inscrit

Résultat := EnsBD $\cup$ EtudBD

2) liste des numéros de personnes qui n'ont rien à voir avec le cours de BD :

réponse : $\prod$ [n°P]Personne – (EnsBD $\cup$ EtudBD)

3) liste des numéros de personnes qui sont enseignants et étudiants simultanément (assistants, doctorants,……)

réponse : ($\prod$ [n°P] etudiant) $\cap$ ($\prod$ [n°P] enseignant)

2.7 Produit cartésien de deux relations n'ayant aucun attribut commun :

Définition : soient deux relations R(A1, A2, …., An) et T(B1, B2,…….Bp) n'ayant pas d'attribut de même nom, le produit de R par T noté : R×T , crée une nouvelle relation temporaire de schéma (A1, A2, …., An, B1, B2,…….Bp), et de population égale à l'ensemble de toutes les concaténations possibles de tuples de R et de T.

Remarque : la différence entre les opérateurs de thêta-jointure et de produit cartésien consiste dans le fait que dans la thêta-jointure, seules les combinaisons des tuples qui satisfont le prédicat de jointure apparaissent dans le résultat, tandis que dans le produit cartésien, toutes les combinaisons de tuples sont retenues.

Exemple : existe-t-il des personnes dont le nom est le même que celui d'un cours ?donner leurs noms :

Réponse : avec un produit et une sélection : $\prod$ *[nom]* σ *[nom = nomC] (Personne* $\times$ *Cours)*

Ou avec une thêta-jointure : $\prod$ *[nom] (Personne* [nom = nomC] Cours*

2.8 La division

Définition : soient deux relations R(A1,……., An) et V(A1,…….., Ap) (p < n), telles que tous les attributs de V sont aussi attributs de R, alors la division de R par V, notée **: R/V**, crée une nouvelle relation temporaire de schéma (A_{p+1}, A_{p+2},…….., An) et de population égale aux tuples de R, tronqués à [A_{p+1}, A_{p+2} ,…….., An], et qui existent dans R, concaténés à tous les tuples de V, c'est à dire :

{< A_{p+1}, A_{p+2} ,…….., An> /$\forall$< A1,…….., Ap>$\in$V, $\exists$< A1,…….., Ap, A_{p+1}, A_{p+2} ,…….., An>$\in$R}

Exemple: liste des étudiants qui peuvent s'inscrire au cours système (c'est à dire qui ont réussi tous les prérequis de ce cours)

Réponse :

Cours prérequis pour système :

ReqSyst := $\prod$ *[nomCprérequis]* σ *[nomC = ''système''] Prérequis)*

Numéros des étudiants qui peuvent s'inscrire au cours de système :

($\prod$ *[nomC, n°E] Obtenu) / (*α *[nomCprérequis : nomC] ReqSyst)*

3. Comment écrire une requête compliquée ?

Exemple : nom des étudiants qui suivent un des cours de l'enseignant numéro 200.

La méthode consiste à représenter visuellement la requête sur le schéma des relations. Elle comprend les étapes suivantes :

1. Identifier les relations utiles pour exprimer la requête.

Personne pour le nom de l'étudiant

Cours pour les cours de l'enseignant numéro 200

Inscrit et étudiant pour faire le lien entre ces deux premières relations.

2. Recopier le schéma de ces relations, et indiquer sur ces schémas :

- Les attributs qui sont partie du résultat de la requête. Dans l'exemple, nom de l'étudiant,
- Les conditions portant sur les attributs. Dans l'exemple, dans Cours, n°Ens=200,
- Les liens entre les relations. Dans l'exemple, n°P de personne = n°P de Etudiant,

  n°E dans Etudiant = n°E dans Inscrit, nomC dans Inscrit = nomC dans Cours.

On obtient donc la figure :

Personne (n°P, nom, adr)
|=
Etudiant (n°p, n°E, dateN
|=
Inscrit (n°E, nomc)
|=
|=

Cours (nomC, cycle, n°Ens)

200

3.Traduire cette figure en expression d'algèbre :

- faire les sélections selon les conditions portant sur les attributs,
- faire les jointures (naturelle ou thêta) selon les liens entre les relations (une jointure par lien),
- projeter sur les attributs qui font partie du résultat.

On obtient ainsi l'expression :

Π [nom] (personne * Etudiant *inscrit *(σ [n°Ens = 200] Cours)

Cette méthode est valable pour la plupart des requêtes. Cependant, certains types de requêtes nécessitent de compliquer la méthode. C 'est le cas des requêtes où la même relation est utilisée plusieurs fois avec des ensembles de tuples différents. Par exemple, liste des noms des étudiants qui habitent dans la même ville que l'étudiant « Ahmed ». ici, la relation Etud doit être représentée par son schéma deux fois, une fois pour « Ahmed » et une fois pour les étudiants recherchés.

Enfin les requêtes comportant l'équivalent sémantique d'un « pour tout » ou d'un « aucun »

Se représentent difficilement visuellement.

Partie 02

Le langage SQL

1.Introduction

Plusieurs langages assertionnels permettant de manipuler des BDDs relationnelles ont été proposés, en particulier QUEL, QBE et SQL. Aujourd'hui, le langage SQL est normalisé et constitue le standard d'accès aux BDDs relationnelles. Les autres interfaces par menus, fenêtres, grilles, etc. ou de programmation type langage de 3ème et 4ème génération, sont le plus souvent offertes au dessus du langage SQL. Celui-ci constitue donc le point d'entrée obligatoire des SGBDs relationnels.

De manière générale, SQL comme tous les autres langages qui ont été proposés utilisent tous des critères de recherche (qualifications) construits à partir de la logique de prédicats du premier ordre. SQL comporte quatre opérations de base :

- La recherche : (mot clé SELECT) permet de retourner des tuples ou parties de tuples.
- L'insertion : (mot clé INSERT) permet d'ajouter des tuples dans une relation.
- La suppression : (mot clé DELETE) permet de supprimer des tuples d'une relation.
- La modification : (mot clé UPDATE) permet de mettre à jour des tuples.

En plus de ces opérations de base, SQL comporte des fonctionnalités de définition de schémas et de contrôle.

2. La recherche de données :

la requête de recherche SELECT est à la base du langage SQL. Elle permet l'interrogation des BDDs relationnelles, en traduisant les opérations de l'algèbre relationnelle.

2.1 Expression de la projection :

Nous rappelons que la projection effectue l'extraction de colonnes (attributs) spécifiés d'une relation, puis élimine les tuples en double. SQL n'élimine pas les doubles, à moins que cela soit explicitement demandé par le mot clé **DISTINCT** ou **UNIQUE** (l'option par défaut est **ALL**). L'expression générale de la projection est :

SELECT [**ALL** / **DISTINCT**] < expression de valeurs>

FROM nom de relation

Expression de valeurs : est une expression arithmétique composée des opérateurs binaires (+, -, *, /), des constantes ou des colonnes.

Une colonne ou un attribut peut être éventuellement précédé d'un nom de relation.

Exemple : soit la BDD composée des relations suivantes :

Produit (Nump, nomp, prix)
Depot (Numdep, libellé, adr-dep)
Stock (Nump, Numdep, qte)

Q1) donner la liste des numéros de produits :

SELECT Nump
FROM Produit

Q2) donner les noms de produits (possibilité d'homonymie)

SELECT DISTINCT Nomp
FROM Produit

Q3) donner la liste (Nump, nomp, prix) des produits avec le prix exprimé en FF

SELECT Nump, nomp, prix*11.5
FROM Produit

2.2 Expression de la sélection :

la sélection s'exprime comme suit :

SELECT *

FROM nom de relation

WHERE qualification (condition de recherche)

Une condition de recherche définit un critère qui est évalué à vrai, faux ou inconnu, selon le résultat de l'application des opérateurs booléens (ET, OU, NOT) à des conditions élémentaires.

Une condition de recherche élémentaire est appelée **prédicat** en SQL. Un **prédicat** permet de comparer deux expressions de valeurs ; la première est appelée terme (contenant des colonnes) et la seconde constante. En SQL , il existe plusieurs prédicats :

- Un prédicat de comparaison utilisant les opérateurs { =, !=, <, >, >=, <= }
- Un prédicat d'intervalle BETWEEN
- Un prédicat de comparaison de texte LIKE

- Un prédicat de test de nullité IS NULL
- Un prédicat d'appartenance IN

Exemples :

Q4) donner la liste des numéros de produits dont le prix est > 1500DA

SELECT Nump
FROM Produit
WHERE prix> 1500

Q5) donner la liste des numéros de produits dont le prix est compris entre 100 et 300 DA et le nom commence par la lettre 'A'

SELECT Nump
FROM Produit
WHERE prix **BETWEEN** 100 **AND** 300
AND nomp **LIKE** A%

Q6) parmi les noms de produits (chaise, table, tableau, stylo, papier), quels sont ceux dont le prix est inconnu ?

SELECT Nump
FROM Produit
WHERE nomp **IN** (chaise, table, tableau, stylo, papier)
AND prix **IS NOT NULL**

2.3 Expression de la jointure et du produit cartésien:

a) Produit cartésien

le produit cartésien s'exprime comme une jointure sans qualification à l'aide de la clause SELECT suivante :

SELECT *

FROM liste de noms de relations

Exemples :

La requête en algèbre relationnelle : P := produit × stock s'exprime en SQL comme suit :

SELECT *
FROM Produit, Stock.

b) jointure:

La jointure avec qualification peut être exprimée en SQL de plusieurs manières

1. En utilisant la restriction du produit cartésien :

SELECT *

FROM liste de noms de relations

WHERE qualification de jointure

3. En utilisant l'imbrication des requêtes.

Remarque : les opérateurs de jointure, de sélection et de projection peuvent être combinés et effectués à l'intérieur d'une même clause SELECT.

Exemples :

Q7) donner le numéro et le pnom des produits en rupture de stock (quantité = 0) :

1ère formulation :

```
SELECT Nump, nomp
FROM Produit, Stock
WHERE  Produit.Nump = Stock.Nump
AND    Stock.qte = 0
```

2ème formulation :

```
SELECT Nump, nomp
FROM Produit
WHERE  Nump IN (SELECT Nump
                FROM Stock
                WHERE  Stock.qte = 0)
```

Remarque:

L'utilisation de l'opérateur d'appartenance (IN) requiert que l'élément dont on veut tester l'appartenance soit du même type que les éléments de l'ensemble.

Q8) donner l'adresse des dépôts qui stockent le produit de nom 'table'

1ère formulation :

```
SELECT Numdep, adr-dep
FROM Produit, Depot, Stock
WHERE  Produit.Nump = Stock.Nump
AND    Depot.Numdep = Stock.Numdep
AND    Produit.nomp LIKE 'table'
```

2ème formulation :

```
SELECT Numdep, adr-dep
FROM Depot
WHERE  Numdep IN (SELECT Numdep
                  FROM Stock
                  WHERE  Nump IN (SELECT Nump
                                  FROM Produit
                                  WHERE  nomp LIKE 'table'))
```

2.4) Expression de l'union, l'intersection et la différence:

Certaines implantations de SQL offrent des opérateurs ensemblistes, qui permettent d'exprimer l'union (UNION), l'intersection (INTERSECT) et la différence (MINUS) à l'aide de la requête :

< clause SELECT> <Opérateur ensembliste> <clause SELECT>

Exemple : soit la BDD : ProduitA (Nump, libellé, prix)

ProduitB (Nump, nom, prix, couleur, poids)

Q9) donner le numéro de tous les produits de la base qui ont un prix > 100DA :

```
SELECT Nump
FROM ProduitA
WHERE prix> 100
   UNION
SELECT Nump
FROM ProduitB
WHERE prix> 100
```

2.5) Utilisation des Alias (synonymes) de relations:

La désignation des attributs d'une relation peut se faire sous l'une des formes suivantes

- Nom-attribut, lorsqu'il n y a pas d'ambiguïté
- Nom-relation.nom-attribut ou Alias-relation.nom-attribut dans le cas de l'existence d'une ambiguïté.

Un alias est un nom de variable attribué à une relation dans la clause FROM afin d 'éviter de répéter le nom complet des relations dans les critères (conditions)

Remarque :

Un attribut sous la 1ère forme (nomattribut) référence la relation la plus interne qui a un attribut de ce nom là.

Exemple :

Q10) donner le libellé des dépôts qui ne stockent pas le produit de numéro 02

```
SELECT  libellé
FROM Depot
WHERE   2 NOT IN (SELECT Nump
                  FROM Stock
                  WHERE     Numdep = Depot.Numdep)
```

Q11) donner les numéros de dépôts qui stockent le même produit stocké dans le dépôt n°= 10 mais en une quantité plus grande :

```
SELECT Numdep
FROM Stock S
WHERE   Nump IN (SELECT Nump
                 FROM Stock
                 WHERE    Numdep = 10
                  AND qte <  S.qte)
```

2.6) Fonctions de calcul et agrégats:

a) fonctions de calcul ou d'agrégation :

Au delà de l'algèbre relationnelle, SQL offre des possibilités de calcul à travers cinq fonctins :

- **COUNT** : permet de compter le nombre de valeurs d'un ensemble.
- **SUM** : permet de sommer les valeurs d'un ensemble.
- **AVG** : permet de calculer la valeur moyenne d'un ensemble
- **MAX** : permet de retrouver la valeur maximale d'un ensemble
- **MIN** : permet de retrouver la valeur minimale d'un ensemble.

Exemples :

Q12) Quel est le nombre de produits stockés dans le dépôt de N°= 10 ?

SELECT COUNT (Nump)
FROM Stock
WHERE Numdep = 10

Q13) Quelle est la quantité totale de stockage du produit de nom 'table' dans les différents dépôts ?

SELECT SUM (qte)
FROM Stock
WHERE Nump **IN** (**SELECT** Nump
FROM Produit
WHERE nomp **LIKE** 'table')

b) Les agrégats

Un agrégat est un partitionnement horizontal d'une table en sous-tables en fonction des valeurs d'un ou de plusieurs attributs de partitionnement suivi, éventuellement, de l'application d'une fonction de calcul aux attributs des sous-tables obtenues.

La forme générale de l'instruction de patitionnement d'une table en groupes est la suivante :

SELECT [Ai1] [,Ai2]….[,Ain], [F1(Aj1)] [,F2(Aj2)] ….[,Fv(Ajv)]

FROM < expression de tables>

[**WHERE** < condition de recherche>]

GROUP BY Ak1 [,Ak2]….[,Akw]

[**HAVING** < condition de groupe >]

avec : Fi = fonction de calcul (COUNT, AVG, SUM, MAX, MIN)

et {Ak1, Ak2,…., Akw} $\supseteq$ {Ai1, Ai2, …., Ain}

et {Ak1, Ak2,…., Akw} $\wedge$ {Aj1, Aj2,…., Ajv} = $\varnothing$

L'ordre d'exécution de cette instruction est le suivant :

1/ Exécuter la clause 'FROM <expression de tables>' pour construire la table de travail définie dans cette clause.

2/ S'il y a une clause 'WHERE', le (s) prédicat (s) est (sont) appliqué (s) à chaque ligne de la table de travail. Les lignes qui valent 'true' sont retenues, les lignes qui valent 'false' ou 'unknown' sont supprimés.

3/ Exécuter la clause 'GROUP BY' qui répartit les lignes dans des groupes où les colonnes de la liste des colonnes spécifiée dans cette clause ont toutes la même valeur. Les 'NULL' sont traités comme s'ils étaient égaux entre eux, et forment donc leur propre groupe.

4/ Appliquer les fonctions de calcul spécifiées dans la clause SELECT sur les groupes générés. Chaque groupe est alors réduit à une seule ligne dans une nouvelle table de résultats.

5/ S'il y a une clause 'HAVING', elle s'applique à tous les groupes. Les groupes pour lesquels le test donne 'true' sont retenus, ceux qui donnent 'false' ou 'unknown' sont supprimés.

Remarque :

La condition de HAVING peut être de deux types :

a) Condition comparant le résultat d'une fonction de calcul portant sur un attribut qui ne fait pad partie de la clause GROUP BY :

F(Ajx) < opérateur de comparaison> Valeur

b) Condition comparant l'ensemble des valeurs prises par un attribut (qui ne fait pas partie de la clause GROUP BY) par les tuples du groupe, cette comparaison se fait en général par rapport à un autre ensemble, à l'aide des opérateurs logiques d'ensemble : =, !=, CONTAINS, NOT CONTAINS. Dans ce cas, on utilise le mot clé 'SET' pour exprimer l'ensemble des valeurs prises par l'attribut pour tous les tuples du groupe.

SET nom-attribut < opérateur de comparaison d'ensemble> <ensemble>

Exemple1 :

Soit la base de données BD2, composée des relations suivantes :

Fournisseur (NF, nomF, ville)

Produits (NP, nomp, poids, couleur)

Produits-Fournisseur (NP, NF, qte)

Q1) Combien de produits sont livrés par chacun des fournisseurs ?

SELECT NF, **COUNT** (NP)

FROM Produits-Fournisseur

GROUP BY NF

Exemple2

Reprenons la BD1:

Q14) Combien de produits sont stockés dans chaque dépôt ?

SELECT Numdep, **COUNT** (Nump)

FROM Stock

GROUP BY Numdep

Q15) Combien de produits sont stockés dans chaque dépôt, tels que la quantité totale des produits stockés dans ce dépôt soit supérieure à 100 ?

SELECT Numdep, **COUNT** (Nump)
FROM Stock
GROUP BY Numdep
HAVING SUM (qte) > 100

Q16) Combien de produits de prix > 150 DA sont stockés dans chaque dépôt ?

```
SELECT Numdep, COUNT (Nump)
FROM  Stock, Produit
WHERE Stock.Nump = Produit.Nump AND prix > 150
GROUP BY  Numdep
```

Q17) Quels sont les numéros de dépôts qui stockent au moins tous les produits qui sont stockés dans le dépôt de numéro 100 ?

```
SELECT Numdep
FROM  Stock
GROUP BY  Numdep
HAVING      SET Nump
            CONTAINS  (SELECT NUMP
                       FROM Stock
                       WHERE Numdep = 100)
```

2.7) Tri des résultats:

La clause ORDER BY permet d'ordonner la relation résultat sur un ou plusieurs attributs. L'ordre peut être croissant (ASC) ou décroissant (DSC). L'ordre ASC est l'ordre par défaut. Si cette clause est présente, elle suit la clause WHERE ou la clause GROUP BY.

Exemple :

Q18) Quelle est la liste des produits qui ne sont pas en rupture de stock triée par ordre croissant des numéros de produits et par ordre décroissant de leur quantité en stock ?

```
SELECT *
FROM Stock
WHERE  qte> 0
ORDER BY Nump ASC, qte DSC
```

Q19) Quelle est la liste des produits avec pour chaque produit, la quantité totale de son stock dans les différents dépôts. Cette liste doit être triée selon les numéros de produits.

```
SELECT nump, SUM(qte)
FROM Stock
GROUP BY Nump
ORDER BY  SUM (qte)
```

Recherche avec des conditions sur des ensembles

En plus des comparateurs logiques d'ensemble (=, !=, CONTAINS, NOT CONTAINS), SQL offre le mot clé **EXISTS (NOT EXISTS)** qui permet de tester si un ensemble est non vide (vide)

```
EXISTS < ensemble>
```

En général, l'ensemble est résultat d'une sous-requête.

Q20) Quels sont les numéros de dépôt qui stockent au moins un produit de prix supérieur à 100 ?

1ère formulation :

```
SELECT DISTINCT   Numdep
FROM Produit, Stock
WHERE   Produit.Nump = Stock.Nump
```

AND Produit.prix > 100

2ème formulation :

```
SELECT  DISTINCT Numdep
FROM Stock
WHERE   Nump IN (SELECT Nump
                 FROM Produit
                 WHERE   prix > 100)
```

3ème formulation :

```
SELECT  DISTINCT Numdep
FROM Stock
WHERE   EXISTS  (SELECT *
                 FROM Produit
                 WHERE   Nump = Stock.Nump
                  AND     prix > 100)
```

2.7) Recherche avec quantificateurs: SOME, ANY, ALL :

SQL propose l'utilisation de sous-requêtes quantifiées par «quel que soit » (ALL), et «il existe » (ANY ou SOME)

Le format général d'une condition élémentaire (prédicat) avec quantificateur est le suivant :

<Valeur-attribut> <opérateur de comparaison> <quantificateur> <ensemble>

avec <quantificateur> ::=SOME, ANY, ALL

ce qui signifie:

- SOME et ANY : existe t-il dans l'ensemble au moins un élément e qui satisfait la condition :

  e <opérateur> <ensemble> ?

- ALL tous les éléments de l'ensemble satisfont-ils la condition ?

Exemple :

Q21) Quels sont les numéros de dépôts qui ne stockent que les produits de prix > 100DA ?

```
 SELECT    Numdep
FROM        Depot
WHERE 100 > ALL (SELECT prix
                 FROM  Produit
                 WHERE  Np = ANY (SELECT Np
                                  FROM Stock
                                  WHERE   Stock.Numdep = Depot.Numdep))
```

3) Mise à jour d'une base de données:

SQL offre trois commandes de mise à jour : INSERT, DELETE et UPDATE. Toute MAJ s'effectue par recherche des tuples à modifier et application des modifications.

3.1) Insertion de tuples :

L'insertion de tuples dans une relation permet de créer de nouvelles lignes. Cette insertion peut s'effectuer de deux manières :

- Par insertion directe en spécifiant les valeurs des attributs du tuple à insérer.
- Via une sous-requête calculant les tuples à insérer.

Syntaxe :

INSERT INTO <nom-relation> [(liste de colonnes)]

{**VALUES** (constante1/expression1, constante2/expression2,….) / <clause SELECT>}

Remarque :

Dans le cas où la liste des colonnes n'est pas spécifiée, tous les attributs de la relation doivent être fournis dans l'ordre de la déclaration. Si seulement certaines colonnes sont spécifiées, les autres sont insérées avec la valeur NULL.

Exemple :

Q22) Insérer le produit de code 'P10' et de nom « écrou » :

```
INSERT INTO   Produit (Nump, nomp)  /* le prix de ce produit est à NULL */
VALUES ("p10", "écrou")
```

3.2) Modification de tuples :

La modification permet de changer des valeurs d'attributs de tuples existants. Elle peut s'effectuer de deux manières :

- Soit par fourniture directe des valeurs à modifier.
- Soit par l'élaboration de ces valeurs à partir d'une sous-requête.

UPDATE <nom-relation>

SET nom-colonne1 = {expression / NULL / clause SELECT} [, colonne2 = ……..]

[**WHERE** <condition de recherche>]

Exemple :

Q23) Mettre à 0 la quantité en stock du produit « P1 » dans tous les dépôts :

```
UPDATE Stock
SET qte = 0
WHERE Nump = "P1"
```

Q24) Mettre à 0 la quantité en stock du produit « P1 » dans tous les dépôts de « Alger » :

```
UPDATE Stock
SET qte = 0
FROM Stock AS S, Depot AS D
WHERE S.Numdep = D.Numdep
AND D.adr = "Alger"
```

Ou bien:

```
UPDATE Stock
SET qte = 0
WHERE Numdep IN  (SELECT    Numdep
                  FROM  Depot
```

WHERE adr = “Alger”)

3.3) suppression de tuples:

la suppression permet d’enlever des tuples d’une relation. La suppression peut concerner :

- tous les tuples de la relation
- un sous-ensemble des tuples de la relation qui vérifie une condition de recherche.

Syntaxe :

DELETE FROM < nom-relation>

[WHERE < condition de recherche>]

remarque : Le schéma d’une relation persiste après la suppression de tous ses tuples car l’opération de suppression opère juste sur le contenu de la relation.

Exemple :

Q25) supprimer tout stockage de produits dans le dépôt « D4 » :

DELETE FROM stock

WHERE numdep = « D4 »

4) Manipulation de schémas de relations

Outre les opérations d’interrogation de BDDs, SQL offre d’autres fonctionnalités de création, suppression et modification de schémas de relations.

4.1) Création de schémas de relation :

SQL permet de créer des schémas de relations et de définir lors de cette création des contraintes d’intégrité sur les attributs. Les contraintes d’intégrité sont des règles ou des assertions qui doivent être vérifiées par les données contenues dans une base. Les contraintes d’intégrité les plus utilisées sont :

a) *Contrainte de non nullité de la valeur d’un attribut :*

Syntaxe : NOT NULL

b) *Contrainte d’unicité d’un attribut:*

Syntaxe : UNIQUE ou PRIMARY KEY

c) *Contrainte référentielle* : elle peut être déclarée pour un attribut ou pour une relation

Syntaxe (attribut) : REFERENCES <table référencée> [<colonne référencée>]

Syntaxe (relation) : [FOREIGN KEY(<colonne référençante>+)]

REFERECES <table référencée> [colonne référencée>+)]

d) *Contrainte générale* : spécifie des plages ou des listes de valeurs possibles

Syntaxe : CHECK <condition>

La commande de création d’un schéma de relation est le suivant :

CREATE TABLE <nom-table> (< élément de table>+)

<élément de table>::= <Définition-colonne> / <contrainte de table>

<Définition-colonne>::= <nom-colonne><type de données>[<clause DEFAULT>][<contrainte-colonne>]

- il existe plusieurs types de données supportés par SQL, par exemple :
  - CHAR (<longueur>)
  - INTEGER
  - REAL……
- La colonne DEFAULT permet de spécifier une valeur par défautselon la syntaxe : DEFAULT < valeur >. La valeur NULL étant permise.

Remarque

Il est possible d'attribuer un nom à une contrainte de la façon suivante :

[CONTRAINT <nom-contrainte>] <contrainte de table / colonne>

Exemple : création de la base de données Stock :

```
CREATE TABLE Produit (
        Nump INT NOT NULL ,
        Nomp  CHAR(15) ,
        Prix REAL  CHECK (prix>0) ,
        PRIMARY KEY (Nump))
CREATE TABLE Depott (
        Numdep INT NOT NULL ,
        libelle  CHAR(20)
         adr CHAR (30)
        PRIMARY KEY (Numdep))
CREATE TABLE Stock (
        Nump INT NOT NULL REFERENCES Produit ,
        Numdep INT NOT NULL ,
        Qte INT ,
        PRIMARY KEY (Nump, Numdep),
        FOREIGN KEY Numdep REFERENCES Depot,
        CHECK (Qte> 0))
```

4.2) Suppression d'un schéma de relation :

la commande qui permet de détruire un schéma de relation est :

DROP TABLE <nom-relation>

Exemple : **DROP TABLE** Stock

4.3) Modification de schémas de relation :

La commande ALTER TABLE permet d'ajouter, supprimer ou modifier des attributs dans un schéma de relation. Cette instruction, qui n'existait pas dans la 1ère version de SQL, varie d'une implémentation à une autre. Elle peut être aussi permise mais sous certaines réserves. Sa syntaxe générale est :

ALTER TABLE <nom-relation> < Action de modification de relation>

<Action de modification de relation> ::=

ADD [COLUMN] <définition de colonne>

| ALTER [COLUMN] <nom-colonne> <action de modification-colonne>

| DROP [COLUMN] <nom-colonne>

| ADD <définition de contrainte-table>

| DROP CONSTRAINT <nom-contrainte>

Exemple :

Dans la relation Stock, ajouter un nouveau attribut « seuil » qui représente le stock minimal d'un produit dans un dépôt avec 'SEUIL'> 0

ALTER TABLE stock

ADD COLUMN Seuil

ADD CHECK (Seuil > 0)

4.4) Définition des vues :

Une vue est une relation virtuelle calculée à partir des relations de base ou à partir d'autres vues à l'aide d'une question (requête). Le schéma de la vue est un schéma externe qui au départ a été utilisé afin de réaliser deux objectifs principaux :

- Garantir une meilleure indépendance logique des programmes par rapport aux données.
- Garantir la sécurité de la base en permettant à l'utilisateur de n'accéder qu'aux données des vues auxquelles il a droit d'accès. Ainsi, les données en dehors de la vue sont protégées.

Actuellement, les vues ont trouvé de nouvelles applications. Par exemple :

- Dans le monde du client-serveur, les vues constituent un élément essentiel d'optimisation des performances.
- Dans le monde des entrepôts de données et du décisionnel, les vues peuvent être concrétisées et permettent de réaliser des synthèses sophistiquées sur les données extraites de la base selon plusieurs dimensions.

La syntaxe générale de la commande SQL de création d'une vue est :

CREATE VIEW <nom-vue> [< liste d'attributs>]

AS < requête>

[WITH CHECK OPTION]

- La clause [WITH CHECK OPTION] permet de spécifier que les tuples insérés ou mis à jour via la vue doivent satisfaire les conditions de la requête définissant la vue.

Exemple :

Créer la vue des produits chers dont le prix est > 100 DA.

CREATE VIEW produits-chers (produit, prix)

AS **SELECT** Nump, prix

FROM produit

WHERE prix> 100

WITH CHECK OPTION