Chap3. - Langages relationnels

Une tâche très importante de tout SGBD est de pouvoir extraire de la base de données des informations souhaitées par l'utilisateur. On parle dans ce cas de langage d'interrogation. On distingue deux grandes classes de langages relationnels :

- Les langages algébriques basés sur l'algèbre relationnelle définie par CODD
- Les langages prédicatifs à variable n-uplet construits à partir de la logique des prédicats.

3.1. Langage algébrique

Ce langage tel qu'il a été définie par CODD dans [ ] est constitué de huit opérateurs, deux groupes de quatre opérateurs chacun.

- L'ensemble traditionnel composé des opérations d'union, intersection, différence et produit cartésien (toutes modifiées quelque peu pour prendre en compte le fait que leurs opérandes sont des relations et pas des ensembles arbitraires).
- Des opérations relationnelles spécifiques de restriction, projection, jointure et division.

Avant de présenter ces opérations, quelques précisions s'imposent.

- Le résultat de chaque opération relationnelle est une autre relation. C'est ce qu'on appelle la propriété de **fermeture** relationnelle : Une requête s'exprime par une relation résultant de l'application successive d'opérateurs algébriques ou relationnels dont les opérandes sont des relations de la base de données.
- En mathématique, l'union de deux ensembles, par exemple, est l'ensemble de tous les éléments appartenant aux deux ou à chacun des ensembles considérés. Comme une relation est un ensemble, informellement (un ensemble de n-uplets), il est évidemment possible de construire l'union de deux relations; le résultat sera un ensemble consistant en tous les n-uplets des deux ou de chacune des relations considérées. Par exemple, l'union de l'ensemble des n-uplets de fournisseurs de la relation FOURNISSEUR et de l'ensemble des n-uplets de pièces de la relation PIECE est certainement possible. Cependant, bien que le résultat soit un ensemble, **ce n'est pas une relation**; les relations ne peuvent contenir un mélange de différents types de n_uplets, elles doivent contenir des n-uplets homogènes. Et bien entendu, nous voulons avoir comme résultat une relation, parce que nous désirons conserver la propriété de fermeture. Si les deux relations ont le même schéma, alors on peut faire notre opération, et le résultat sera également une relation de même schéma; en d'autres termes la propriété de fermeture est préservée.

3.1.1. Opérateurs ensemblistes

- **Union**

L'union de deux relations de même schéma R1 et R2 notée **R1 ∪ R2** est une relation ayant le même schéma que R1 et R2 et une structure consistant en l'ensemble de tous les n-uplets *t* appartenant à R1, à R2 ou aux deux.

- **Intersection**

L'intersection de deux relations de même schéma R1 et R2 notée **R1 ∩ R2** est une relation ayant le même schéma que R1 et R2 et une structure consistant en l'ensemble de tous les n-uplets *t* appartenant à R1 et R2.

- **Différence**

La différence de deux relations de même schéma R1 et R2, dans cet ordre, notée **R1 - R2** est une relation ayant le même schéma que R1 et R2 et une structure consistant en l'ensemble de tous les n-uplets *t* appartenant à R1 et pas à R2

Exemple :

Soit R1(A, B, C) et R2(A, B, C) dont les extensions sont les suivantes :

R1

A	B
a	b
c	b
d	e

R2

A	B
a	b
c	e
d	b

R1 ∪ R2 = R

R

A	B
a	b
c	b
d	e
c	e
d	b

R1 ∩ R2 = R

R

A	B
a	b

R1 - R2 = R

R

A	B
c	b
d	e

Observons que R1 ∩ R2 = R1 - (R1- R2)

- **Produit cartésien**

Le produit cartésien de deux relations R1 et R2 noté **R1 ⊗ R2** est une relation ayant pour attributs la concaténation de ceux de R1 et de R2 et dont les n-uplets sont toutes les concaténations d'un n-uplet de R1 à un n-uplet de R2.

Remarque : dans le cas où les deux relations possèdent des noms d'attributs communs, le SGBD procède à renommer les attributs en les faisant précéder du nom de leur relation

Exemple:

Soit la relation R3(B, C) dont l'extension est la suivante :

R3

B	C
b	c
a	a
b	d

R1 ⊗ R3 = R

R

A	R1.B	R3.B	C
a	b	b	c
a	b	e	a
a	b	b	d
c	b	b	c
c	b	e	a
c	b	b	d
d	e	b	c
d	e	e	a
d	e	b	d

3.1.2. Opérateurs relationnels spécifiques

- **Restriction**

La restriction d'une relation de schéma R(A1, A2, … , An) sous une certaine condition C est une relation R' de même schéma que R dont les n-uplets sont un sous ensemble de R vérifiant la condition C. Cette opération est notée :

$\sigma_C(R)$

La condition C est exprimée comme une combinaison logique de termes. Chaque terme est une simple comparaison de type Ai θ *littéral* ou bien Ai θ Aj où Ai et Aj sont des attributs et θ est un opérateur quelconque de comparaison (=, ≠, <, >, etc.). Le résultat de cette comparaison sera vrai ou faux.

Exemples :

Soit l'expression de restriction

$\sigma_{VILLE = Alger}$ (FOURNISSEUR)

Le résultat de cette expression est :

NF	NOM	CODE	VILLE
F 1	Haroun	120	Alger
F 4	Kaci	120	Alger

C'est une relation de même schéma que la relation FOURNISSEUR qui donne toutes les informations sur les fournisseurs d'Alger.

L'expression suivante :

$\sigma_{NOM = Ecrou \text{ and } MATERIAU = Fer}$ (PIECE)
a pour résultat :

N P	NOM	MATERIAU	POIDS	VILLE
P 4	Ecrou	Fer	14	Alger

C'est une relation ayant le même schéma que la relation PIECE qui donne toutes les informations sur les pièces dont le nom est écrou et le matériau est fer.

- **Projection**

La projection d'une relation R de schéma R(A1, A2, … , An) sur les attributs Ai_1, Ai_2, … Ai_p avec $i_j \neq i_k$ et $p<n$ est une relation R' de schéma R'(Ai_1, Ai_2, … Ai_p) dont les n-uplets sont obtenus par élimination des valeurs des attributs de R n'appartenant pas à R' et par suppression des n-uplets en double. Cette opération est notée :

$\pi_{Ai1, Ai2, ...Aip}$ (R)

A l'instar de la restriction, l'opérateur de projection prend une seule relation comme argument, ainsi qu'un paramètre, qui est la liste des attributs choisis parmi ceux du schéma de la relation donnée en argument.
C'est donc un opérateur pour la construction "verticale" d'un sous-ensemble de la relation R

Exemples :

π_{Ville} (FOURNISSEUR)

VILLE
Alger
Tunis
Paris

C'est une relation qui donne les villes où sont localisés les fournisseurs.

$\pi_{Code}(\sigma_{VILLE = Alger}(FOURNISSEUR))$

CODE
120

C'est une relation qui donne le code des fournisseurs localisés à Alger. Ce résultat est obtenu par application successive de l'opération de restriction suivie d'une opération de projection.

- **Jointure naturelle**

Etant donné deux relations R1(A1, … An, X) et R2(X, B1, … , Bm) ayant un ensemble X d'attributs communs (définis sur le même domaine), la jointure naturelle de ces deux relations R1 et R2 notée ∞ est une relation R(A1, … An, X, B1, …, Bn) tel que tout n-uplet t résultant de la composition d'un n-uplet de R1 et d'un n-uplet de R2 ayant les mêmes valeurs pour les attributs X appartient à R. En d'autres termes si t1 est un n-uplets de R1 et t2 est un n-uplet de R2 et t1[X] = t2[X] alors

$$R = R1 \infty R2 = \{ t / t = t1 \cup X \cup t2 : (t1 \cup X \in R1) , (X \cup t2 \in R2) \}$$

Remarques :

La jointure telle que nous l'avons défini est à la fois associative et commutative. Comme conséquence, les expressions :

(R1 ∞ R2) ∞ R3

et

R1 ∞ (R2 ∞ R3)

Peuvent toutes les deux être simplifiées sans ambiguïté en

R1 ∞ R2 ∞ R3

De même, les expressions :

R1 ∞ R2

et

R2 ∞ R1

sont équivalentes

Remarquons aussi que si R1 et R2 n'ont pas de nom d'attribut commun, alors R1 ∞ R2 est équivalent à R1 ⊗ R2 (la jointure naturelle dégénère en un produit cartésien dans ce cas).

Exemple :

PIECE ∞ π_{NF}(FOURNITURE)

N P	NOM	MATERIAU	POIDS	VILLE	NF
P 1	Vis	Fer	12	Alger	F1
P1	Vis	Fer	12	Alger	F2
P2	Boulon	Acier	17	Tunis	F1
P 2	Boulon	Acier	17	Tunis	F2
P2	Boulon	Acier	17	Tunis	F3
P 3	Ecrou	Zinc	17	Paris	F1
P4	Ecrou	Fer	14	Alger	F1
P 4	Ecrou	Fer	14	Alger	F4
P5	Came	Zinc	12	Tunis	F1
P 5	Came	Zinc	12	Tunis	F4
P6	Clou	Fer	19	Alger	F1

C'est une relation qui donne les informations sur les pièces ainsi que le numéro des fournisseurs qui les fournissent. C'est le résultat d'une jointure entre PIECE et FOURNITURE sur l'attribut commun NF.
Pour savoir où sont stockées les pièces fournies par le fournisseur F1, on écrit :

π_{Ville}(PIECE ∞ $\sigma_{NF = F1}$(FOURNITURE))

et on obtient :

VILLE
Alger
Tunis
Paris

Si on veut obtenir le nom des pièces fournies par les fournisseurs localisés à Alger, on écrit :

π_{NOM}(($\sigma_{Ville = Alger}$(FOURNISSEUR)) ∞ FOURNITURE) ∞ PIECE)

On obtient :

NOM
Vis
Ecrou
Clou

- **θ-jointure**

Etant donné deux relations R1(A1, …, An, X) et R2(Y, B1, …, Bm) et une condition de la forme X θ Y, la θ-jointure de R1 et R2 notée :

R1 ⊗ R2 WHERE X θ Y

est la restriction du produit cartésien à cette condition.

Cette opération est relativement rare, c'est une jointure de deux relations sur la base d'une condition autre que l'égalité. C'est donc une relation possédant le même en-tête que le produit cartésien de R1 et R2 et ayant une structure consistant en l'ensemble de tous les n-uplets *t* appartenant à ce produit cartésien avec la condition "X θ Y" évaluée à vrai pour ce n-uplet *t*. Les attributs X et Y doivent être définis sur le même domaine et l'opération doit avoir un sens pour ce domaine. Par exemple si l'on souhaite calculer la *jointure supérieur-à* de la relation FOURNISSEUR sur ville avec la relation PIECE sur Ville (nous supposons que ">" à un sens pour les villes, et s'interprète comme "supérieur dans l'ordre alphabétique"), on écrit :

FOURNISSEUR ⊗ PIECE WHERE FOURNISSEUR.VILLE > PIECE.VILLE

On obtient :

NF	F.NOM	CODE	F.VILLE	NP	P.NOM	MATERIAU	POIDS	P.VILLE
F2	Bouzid	110	Tunis	P1	Vis	FER	12	Alger
F2	Bouzid	110	Tunis	P3	Ecrou	Zinc	17	Paris
F2	Bouzid	110	Tunis	P4	Ecrou	Fer	14	Alger
F2	Bouzid	110	Tunis	P6	Clou	Fer	19	Alger
F3	Mamir	130	Tunis	P1	Vis	FER	12	Alger
F3	Mamir	130	Tunis	P3	Ecrou	Zinc	17	Paris
F3	Mamir	130	Tunis	P4	Ecrou	Fer	14	Alger
F3	Mamir	130	Tunis	P6	Clou	Fer	19	Alger
F5	Kaddour	130	Oran	P1	Vis	FER	12	Alger
F5	Kaddour	130	Oran	P4	Ecrou	Fer	14	Alger
F5	Kaddour	130	Oran	P6	Clou	Fer	19	Alger

- **Division**

La division est définie comme une opération entre une relation binaire (le dividende) et une relation unaire le diviseur qui produit une relation unaire le quotient comme résultat.
Soit R(X,Y) et S(Y) où X = A1, …, An et Y = B1, …, Bm. L'opération de division

R ÷ S

produit un quotient qui est une relation Q(X) : un n-uplet *t*X n'apparaît dans le quotient que si la paire *t*.< X,Y> apparaît dans R pour toutes les valeurs *t*.Y apparaissant dans S. En d'autres termes, le quotient de la relation R(X, Y) par une relation S(Y) est une relation Q(X) tel que chaque n-uplet de Q concaténé avec chaque n-uplet de S donne un n-uplet de R.
Exemples :

Quels sont les fournisseurs qui fournissent toutes les pièces? S'écrit :

$\pi_{NF,NP}$(FOURNITURE) ÷ π_{NP}(PIECE)

On obtient :

NF
F1

Dans notre exemple il n'y a que F1 qui répond à cette contrainte i.e. fournir toutes les pièces.

Quelles sont les pièces fournies par tous les fournisseurs? S'écrit :

$\pi_{NF, NP}$**(FOURNITURE)** $\div$ π_{NF}**(FOURNISSEUR)**

On obtient : $\varnothing$

En effet, dans notre exemple le fournisseur F5 ne fournit aucune pièce, nous ne pouvons donc pas avoir de pièces qui sont fournies par tous les fournisseurs.

Nous remarquons qu'à chaque fois que la requête formulée en langue naturelle comporte une condition faisant intervenir le "tou(te)s" alors ceci est une indication forte que la division est l'opération à effectuer.

Important! L'opérateur de division comporte quelques pièges; plus particulièrement, des difficultés peuvent apparaître liées aux relations vides.

3.1.3. Exemples

1. Obtenir le nom des fournisseurs de la pièce P2

$\pi_{NOM}(\sigma_{NP = P2}$**(FOURNITURE)** ∞ **FOURNISSEUR)**

2. Obtenir le nom des fournisseurs qui fournissent au moins une pièce en Fer

π_{NOM}**[FOURNISSEUR** ∞ **(** $\sigma_{MATERIAU = Fer}$ **(PIECE)** ∞ **FOURNITURE)]**

3. Obtenir le nom des fournisseurs de toutes les pièces

π_{NOM}**(FOURNISSEUR** ∞ **(**$\pi_{NF,NP}$**(FOURNITURE)** $\div$ π_{NP}**(PIECE)))**

4. Obtenir le numéro des fournisseurs d'au moins toutes les pièces fournies par F2

$\pi_{NF,NP}$**(FOURNITURE)** $\div$ $\pi_{NP}(\sigma_{NF = F2}$ **(FOURNITURE))**

5. Obtenir le numéro des fournisseurs qui fournissent que toutes les pièces fournies par F2

[$\pi_{NF,NP}$**(FOURNITURE)** $\div$ $\pi_{NP}(\sigma_{NF = F2}$ **(FOURNITURE))] - …..**

Obtenir le nom des fournisseurs qui ne fournissent pas la pièce P2

π_{NOM}[FOURNISSEUR ∞ (π_{NF} (FOURNISSEUR) - π_{NF} ($\sigma_{NP = P2}$ (FOURNITURE)))]

3.1.4. Opérations de mise à jour

En plus des opérateurs ensemblistes et relationnelles que nous venons de voir nous utilisons l'opérateur d'affectation noté := dans les opérations de mise à jour. L'opérateur d'affectation fait en sorte de "mémoriser" la valeur de certaines expressions algébriques dans la base de données.

- **Insertion**

L'insertion n'est autre qu'une opération d'union d'un n-uplet avec l'ensemble des n-uplets d'une relation.

- **Suppression**

La suppression n'est autre qu'une opération de différence d'une relation R avec le n-uplet à supprimer.

- **Modification**

La modification n'est autre que la combinaison d'une opération de différence (suppression) suivie d'une union (insertion).

Exemples :

Insérer la pièce (NP 'P7', NOM 'Rondelle', MATERIAU 'Bronze', POIDS '4', VILLE 'ALGER'). S'écrit :

PIECE : = PIECE $\cup$ {('P7' , 'Rondelle' , 'Bronze' , '4' , 'ALGER')}

Supprimer le fournisseur F1. S'écrit :

FOURNISSEUR : = FOURNISSEUR - { ('F1', ?, ?, ?)

Remarque :

L'utilisation de $\cup$ et de - en remplacement des opérations explicites INSERT et DELETE n'est pas vraiment satisfaisante parce que $\cup$ et - ne traitent pas convenablement les situations d'erreur, i.e., ces opérateurs n'arrivent pas à traiter les situations telles que : rejeter une tentative d'effacement d'un n-uplet qui n'existe pas.
Dans la pratique, les systèmes relationnels proposent des opérations d'insertion et de suppression plus élaborées.

3.2 Langage Prédicatif

Le langage prédicatif est fondé sur le calcul des prédicats du premier ordre. Il est intéressant de faire quelques rappels sur la logique du premier ordre.

3.2.1. Rappels sur la logique du premier ordre.

La logique du premier ordre est avant tout un langage avec sa syntaxe et son interprétation.

- **La syntaxe du langage comporte :**

1. La spécification d'un alphabet de symboles,
2. La définition de termes et d'expressions utiles appelées formules bien formées (ou formules) qui peuvent être construites à partir de ces symboles.

Le vocabulaire de base comporte l'ensemble des symboles suivants :
1. Les parenthèses (,)
2. Un ensemble de constantes notées : a, b, c, …
3. Un ensemble de variables notées : x, y, z, …
4. Un ensemble de prédicats à n arguments notés : P, Q, R, …
5. Un ensemble de fonctions à m arguments notées : f, g, h, …
6. Un ensemble de connecteurs logiques : $\rightarrow$ (implication matérielle), $\neg$ (négation), $\wedge$ (et logique), $\vee$ (ou logique)
7. Le quantificateur existentiel $\exists$ et quantitatif $\forall$

A l'aide de ce vocabulaire, on fabrique des termes de la façon suivante :
1. Les symboles de constantes et de variables sont des termes,
2. Si f est un symbole de fonction et t1, t2, t3, …, tn sont des termes alors f(t1, t2, …, tn) est un terme.

On construit ensuite les formules qui sont des phrases du langage de la façon suivante :
1. Un prédicat se définit en considérant un ensemble X et une variable x qui parcourt les éléments de X. On appelle prédicat P(x) une forme d'énoncé qui devient une proposition pour chaque valeur affectée à x, i.e., que son évaluation conduit à lui attribuer la notion de vrai ou de faux.

Une formule atomique est définie comme un prédicat à m arguments, dont les arguments sont des termes : si P est un prédicat et t1, t2, …, tm sont des termes alors P(t1, t2, …., tm) est une formule atomique.

Une formule atomique est une formule bien formée (fbf).
- Si f1 et f2 sont des fbfs alors f1 $\wedge$ f2, f1 $\vee$ f2, $\neg$f1, f1 $\rightarrow$ f2 sont des fbfs
- Si f1 est une fbf alors $\exists$ xf1, $\forall$x f1 sont des fbfs
- (f1) est une fbf

Une variable est libre dans une formule si elle n'est soumise à aucun quantificateur.
Une formule est fermée si toutes les variables sont quantifiées sinon elle est ouverte.

Quelques équivalences logiques.

- $\neg\neg P \equiv P$
- $P \vee Q \equiv \neg(\neg P \wedge \neg Q)$
- $P \wedge Q \equiv \neg(\neg P \vee Q)$
- $P \rightarrow Q \equiv \neg P \vee Q$ et se lit si P alors Q
- $\forall x\, P(x) \equiv \neg\exists x \neg P(x)$
- $\forall x\, P(x)$ signifie " toute chose a la propriété P
- $\exists x\, P(x)$ signifie " il y a au moins une chose qui a la propriété P "

- **Interprétation**

La syntaxe du langage étant définie, il s'agit de donner une signification aux formules. Cela revient à les interpréter comme des assertions sur le domaine de discours.

Dans les bases de données relationnelles, le domaine de discours est l'ensemble des occurrences de la base, i.e., l'ensemble des domaines des attributs des relations.

La spécification du domaine de discours D conduit à préciser une interprétation des fbfs comme suit :

- Toute constante doit être associée à un élément particulier de l'ensemble des occurrences de la base de données.
- Toute variable doit être associée à un ensemble d'éléments de D, i.e., cet ensemble doit regrouper des éléments de même nature
- Tout prédicat doit être associé à des relations particulières des éléments de D. Un prédicat n-aire est une fonction de

$$D^n \rightarrow \{\text{Vrai, Faux}\}$$

Etant donné une fbf et une interprétation nous pouvons assigner une valeur Vraie ou Fausse à chaque formule atomique composant la fbf. Il suffit pour cela de faire correspondre au prédicat la valeur Vraie si les éléments du domaine de discours satisfont la relation et faux sinon.

Deux grandes classes de langages prédicatifs ont été développées qui se distinguent essentiellement par l'interprétation des variables.

- les langages prédicatifs à variables n-uplets dont les variables sont interprétées comme des n-uplets de la base de données,
- les langages prédicatifs à variables domaines dont les variables sont interprétées comme des domaines des attributs de la base de données.

Dans ce cours nous nous intéressons qu'au langage prédicatif à variables n-uplets. Pour le deuxième type de langage prédicatif nous renvoyons le lecteur à d'autres ouvrages spécialisés en base de données.

3.2.2. Le calcul relationnel à variables n-uplets

Définition

Le calcul relationnel à variables n-uplets se déduit du calcul des prédicats en interprétant les formules bien formées comme suit :

1. Les variables sont associées à des n-uplets

2. Les seuls termes considérés sont les constantes associées aux valeurs des domaines des attributs, par exemple "rouge", et les fonctions génératrices d'attributs notées v.A où v est une variable et A un attribut (par exemple v.couleur).
3. Les prédicats utilisés sont ceux permettant les comparaisons logiques, i. e., =, ≠, <,>, ≤, ≥.

Il est possible de définir la syntaxe du calcul relationnel à variables n-uplets sous la Forme Normal de Backus (B. N. F.) comme suit :

Requête ::= { Projection / Formule }

Formule ::= Defvar / Defvar Λ Qualification

Defvar ::= Relation(variable) / Defvar Λ Relation(variable)

Qualification ::= Terme θ Terme / Terme θ Constante / Qualification V Qualification / ¬ Qualification / ∃ Variable Qualification / ∀ Variable Qualification

Terme ::= Variable.Attribut

Projection ::= Terme / projection , Terme

θ ::= = / ≠ / < / > / ≤ / ≥.

Variable ::= A / B / … / Z

Note : toute variable apparaissant dans le critère de projection doit être libre dans la formule (non qualifiée).

Si nous voulons répondre à la question :

1. Quels sont les fournisseurs (description complète) localisés à Alger?

Nous avons vu qu'un n-uplet est une ligne dans une table. Une table étant considérée comme un ensemble de n-uplets, nous pouvons donc décrire la table Fournisseur de la façon suivante:

{ f / f ∈ Fournisseur }

i.e., l'ensemble des n-uplets f tels que "f ∈ Fournisseur" est vrai, en utilisant la forme logique suivante :

{ f / P(f) } où P est le prédicat Fournisseur.

Mais notre requête ne répond pas à la question puisque nous avons tous les n-uplets de la table. Il est nécessaire de définir des conditions plus contraignantes : nous désirons les nuplets de la table fournisseurs pour lesquels la valeur prise par l'attribut Ville est égale à "Alger"

{ f / Fournisseur(f) ∧ f.Ville = "Alger" }

2. Si nous voulons la liste des noms et des matériaux de toutes les pièces, nous écrivons :

{ p.nom, p.matériau / Pièce(p) }

3. Donner pour chaque nom de pièce le numéro de fournisseur associé, s'écrit :

{ p.Nomp, x.NF / Pièce(p) Λ Fourniture (x) Λ p.NP = x.NP }

En utilisant le quantificateur existentiel:

4. Donner le nom des fournisseurs ayant fournis au moins une pièce en fer, s'écrit :

{ f.Nom / ∃p ∃x Fournisseur(f) Λ Fourniture(x) Λ Pièce(p) Λ f.NF = x.NF Λ x.NP = p.NP Λ p.Matériau = ''fer''}

En fait toute variable apparaissant dans la formule mais non dans la liste résultat est qualifiée par "∃".

En utilisant le quantificateur universel

5. Donner le nom des fournisseurs qui ne fournissent pas p1, s'écrit :

{ f.Nom / Fournisseur(f) Λ ∀ x Fourniture(x) Λ f.NF ≠ x.NF V x.NP ≠ P1 }

Sachant que ∀x P(x) ≡ ¬∃ x¬P(x) alors la requête précédente peut se réécrire en :

{ f.Nom / Fournisseur(f) Λ ¬∃ x Fourniture(x) Λ f.NF = x.NF Λ x.NP = P1 }

En utilisant les deux quantificateurs

6. Donner le nom des fournisseurs ayant fournis toutes les pièces :

{f.Nom / Fournisseur(f) Λ ∀y pièce (y) ∃x Fourniture(x) Λ f.NF = x.NF Λ x.NP = y.NP}

signifie : pour toutes les pièces il existe un n-uplets x de fourniture reliant le numéro de fournisseur et le numéro de pièce.

En utilisant l'implication :

7. Donner le numéro des fournisseurs qui fournissent au moins toutes les pièces fournies par le fournisseur F2.

{f.NF / Fournisseur(f) Λ ∀p (pièce (p) ∃x (Fourniture(x) Λ x.NF = F2 Λ x.NP = p.NP) → ∃y (Fourniture(y) Λ y.NF = f. Λ y.NP = p.NP))}

Le symbole → représente l'implication et notre requête peut être lue : " pour toutes les pièces, si F2 fournit la pièce, alors le fournisseur f fournie la pièce".

L'implication est intuitivement plus facile mais on peut toujours écrire la requête en remplaçant $A \rightarrow B$ par $(\neg A) \vee B$

$$\{f.NF / Fournisseur(f) \wedge \neg\exists p\ (pièce\ (p) \wedge \exists x\ (Fourniture(x) \wedge x.NF = F2 \wedge x.NP = p.NP) \wedge \neg\exists y\ (Fourniture(y) \wedge y.NF = f. \wedge y.NP = p.NP))\}$$

3.3. Le langage SQL

Le langage SQL (Structured Query Language) a été développé dans les laboratoires d'IBM pour le système relationnel expérimental SYSTEM-R, durant les années 70. Par la suite, il est devenu un standard pour les SGBD relationnels. Il comporte quatre groupes d'instructions:

1. les instructions d'interrogation de données,
2. les instructions de définition de données pour la création de tables, d'introduction de nouveaux attributs dans la les tables existantes et la création des index,
3. les instructions de manipulation des données par l'insertion, la suppression et la modification des données,
4. les instructions de contrôles pour la mise en place et la suppression des autorisations d'accès aux données de la base.

Nous nous intéressons pour l'instant au premier type d'instructions, i.e., l'interrogation des données. Nous verrons les autres types d'instructions au niveau du chapitre quatre.

Une requête d'interrogation est représentée syntaxiquement comme un bloc de qualification :

Select <liste d'attributs>
From < nom-relation>
[**Where** <prédicat>]
[**Group By** <attribut>]
[**Having** <pedicat>]
[**Order By** <attribut>[**Desc**]]**;**

La forme la plus simple du Select

Select <liste d'attributs>
From < nom-relation>**;**

Permet d'extraire de la table une sous-table obtenue par projection selon le (ou les) attribut(s) précisé(s). Le Select a donc le sens de l'opérateur Project

L'option * (astérix) à la place du nom des attributs permet l'édition de la table dans sa totalité.

Exemple:
1. Donner le numéro et le code des fournisseurs d'Alger, s'écrit :

Select NF, CODE
From Fournisseur
Where Ville = 'Alger'**;**

2. Donner toutes les informations sur les fournisseurs s'écrit :

Select *
From Fournisseur;

Lors de l'opération de projection, SQL n'élimine pas les doubles dans la réponse à une interrogation. Si l'utilisateur désire le faire, il doit l'exprimer explicitement à l'aide du mot clé **Unique.** L'élimination des n-uplets redondants est une opération très coûteuse, aussi SQL laisse la liberté d'utilisation de ce mot clé aux utilisateurs.

Exemple :

3. Donner toutes les villes où résident des fournisseurs s'écrit :

Select Unique VILLE
From Fournisseur;

Le prédicat de la clause **Where** peut inclure les opérateurs de comparaison : =, ≠, <, >, ≤, ≥ et les opérateurs booléens **And**, **Or**, **Not.**
D'autres opérateurs ont été introduits dans les dernières versions d'Oracles, nous citons :

- **Between :** exp1 **Between** expr2 **And** exp3 vrai si exp1 est comprise entre exp2 et exp3, faux sinon.
- **IN :** exp1 IN (exp2, exp3, …) vrai si exp1 est égale à l'une des expressions de la liste entre parenthèses.
- **LIKE :** exp LIKE <chaîne> où chaîne est une chaîne de caractère pouvant contenir un caractère joker ′-′ qui remplace un caractère quelconque ou bien ′%′ qui remplace une chaîne de longueur quelconque.

Exemples :

1. Donner le nom des fournisseurs dont le code est compris entre 15 et 30

```
SELECT NOM
FROM FOURNISSEU
WHERE CODE BETWEEN 120 AND 130;
```

2. Sélectionner les pièces qui sont en fer ou en zinc :

```
SELECT NOM
FROM PIECE
WHERE MATERIAU IN (′fer′, ′zinc′);
```

3. Donner le nom des fournisseurs dont le nom commence par la lettre ′M′ :

```
SELECT NOM
FROM FOURNISSEUR
WHERE NOM LIKE  ′M%′;
```

4. Donner le nom des fournisseurs dont le nom comprend la lettre L en deuxième position :

```
SELECT NOM
FROM FOURNISSEUR
WHERE NOM LIKE  '-L%';
```

5. Donner le nom des fournisseurs d'Alger dont le code est supérieur à 120 :

```
SELECT NOM
FROM FOURNISSEUR
WHERE VILLE = 'Alger' AND Code > 120;
```

Le tri du résultat d'un SELECT

Le résultat d'un select peut être trié dans l'ordre ascendant ou descendant en fonction d'une ou plusieurs colonnes. Les critères du tri sont spécifiés par la clause ORDER BY dont la syntaxe est :

ORDER BY <attribut1> [DESC][,<attribut2> [DESC],…]

Le tri se fait d'abord selon la première colonne spécifiée dans l'order by, puis les lignes ayant la même valeur dans la première colonne sont triées selon la deuxième colonne de l'order by etc.…Le tri peut être ascendant ou descendant pour chaque colonne. Par défaut, il est ascendant.

Exemple :

```
SELECT VILLE NOM
FROM FOURNISSEUR
ORDER BY VILLE;
```

Opération de jointure

La jointure en SQL s'exprime en imbriquant des blocs de sélection et en utilisant des opérateurs d'appartenance (IN, NOT IN) (CONTAINS, NOT CONTAINS),…

Exemple :

Donner le nom des fournisseurs qui fournissent la pièce P2.

```
SELECT NOM
FROM FOURNISSEUR
WHERE NF IN SELECT NF
                FROM FOURNITURE
                WHERE NP = P2;
```

Parfois, nous avons besoin d'utiliser une ''référence'' inter-block dans une requête SQL pour répondre à une question telle que :

Donner le nom des fournisseurs qui ne fournissent pas P1

```
SELECT NOM
FROM FOURNISSEUR
WHERE ′P1′ NOT IN SELECT NP
                    FROM FOURNITURE
                    WHERE NF = FOURNISSEUR.NF;
```

Attention une requête SQL du type suivant ne répond pas à la question :

```
SELECT NOM
FROM FOURNISSEUR
WHERE NF IN SELECT NF
                    FROM FOURNITURE
                    WHERE NP ≠ ′P1′ ;
```

Car un fournisseur produisant ′P1′ peut être sélectionné.

Nous pouvons imbriquer des blocs SELECT en faisant intervenir la même relation.

Exemple :

Donner le numérob de fournisseurs fournissant au moins une pièce fournie par le fournisseur ′F2′, s'écrit

```
SELECT UNIQUE NF
FROM FOURNITURE
WHERE NP IN SELECT NP
                    FROM FOURNITURE
                    WHERE NF = ′F2′;
```

Il est parfois nécessaire d'introduire les variables synonymes pour lever certaines ambiguïtés.

Exemple :

Donner le numéro des pièces fournies par plus d'un fournisseur.

```
SELECT UNIQUE NP
FROM FOURNITURE   x
WHERE NP IN SELECT NP
                    FROM FOURNITURE
                    WHERE NF ≠ x.NF;
```

Une autre manière d'exprimer la jointure en SQL est illustrée par l'exemple suivant :

Exemple :
Pour chaque pièce, donner son numéro et le nom des villes des fournisseurs fournissant cette pièce.

```
SELECT UNIQUE NP
FROM FOURNITURE, FOURNISSEUR
WHERE FOURNITURE ? NF = FOURNISSEUR.NF;
```

Le prédicat **EXISTS** contient une sous-interrogation qui, associée à la condition du EXISTS, peut être évaluée comme vraie ou fausse. Le prédicat EXISTS peut être utilisé à chaque fois qu'une interrogation employant le prédicat IN peut être utilisée. Il est également possible de formuler une interrogation à l'aide de **NOT EXISTS**.

Exemples :
Donner le nom des fournisseurs fournissant la pièce ′P2′.

```
SELECT NOM
FROM FOURNISSEUR
WHERE EXISTS (SELECT *
                    FROM FOURNITURE
                    NF = FOURNISSEUR.NF AND NP = ′P2′);
```

Donner le nom des fournisseurs ne fournissant pas ′P2′.

```
SELECT NOM
FROM FOURNISSEUR
WHERE NOT EXISTS (SELECT *
                          FROM FOURNITURE
                          WHERE NF = FOURNISSEUR.NF AND NP = ′P2′ ;
```

Expression de la division

Pour exprimer la division, SQL utilise une recherche avec NOT EXIST

Exemple :

```
SELECT NOM
FROM FOURNISSEUR
WHERE NOT EXISTS (SELECT *
                          FROM PIECE
                          WHERE NOT EXISTS (SELECT *
                                                  FROM FOURNITURE
                                                  WHERE NF = FOURNISSEUR .NF
                                                   AND NP = PIECE.NP));
```

Cette requête peut être paraphrasée en : donner le nom des fournisseurs tels qu'il n'existe pas de pièces qu'ils ne fournissent pas.

UNION, INTERSECT et MINUS

SQL utilise les opérateurs ensemblistes UNION pour l'union, INTERSECT pour l'intersection et MINUS pour le moins. Ces opérateurs permettent, comme dans le langage algébrique de ″joindre″ des tables verticalement, i.e., de combiner dans un résultat unique des lignes provenant de deux interrogations de la manière suivante :

```
SELECT …..
UNION/INTERSECT/MINUS
SELECT….
```

Exemple :

Donner le numéro des fournisseurs qui ne fournissent aucune pièce.

```
SELECT NF
FROM FOURNISSEUR
MINUS
SELECT  NF
FROM FOURNITURE;
```

Remarques :

Dans une interrogation utilisant des opérateurs ensemblistes :

1. tous les SELECT doivent avoir le même nombre de colonnes sélectionnées, et leurs types doivent être un à un identiques.
2. les doublons sont éliminés, i.e., l'option UNIQUE/DISTINCT est implicite.
3. les noms des colonnes titres sont ceux de la première relation
4. nous pouvons combiner le résultat de plus de deux SELECT au moyen de ces opérateurs, i.e., SELECT1 UNION SELECT2 INTERSECT SELECT 3 ….Dans ce cas l'expression sera évaluée en combinant les deux premiers SELECT à partir de la gauche puis le résultat avec le troisième SELECT etc. Comme dans l'expression arithmétique, nous pouvons modifier l'ordre d'évaluation en utilisant les parenthèses

Exemple :

SELECT1…UNION (SELECT2… INTERSECT SELECT3…)

Opérateurs d'évaluation d'ensemble

Comme beaucoup de langages relationnels, SQL offre la possibilité d'utiliser sur un ensemble d'informations les opérateurs :

COUNT : pour le dénombrement des éléments
SUM : pour la somme des éléments
AVG : pour la moyenne des éléments
MAX : pour la recherche du maximum
MIN : pour la recherche du minimum

Exemple :

Donner le nombre des fournisseurs

```
SELECT COUNT(NF)
FROM FOURNISSEUR;
```

Donner la quantité totale des pièces p fournies

```
SELECT SUM(QTE)
FROM FOURNITURE
WHERE NP = ′P2′ ;
```

Attention ! Ce résultat est d'un type différent de ceux vus précédemment. Il est, par exemple, impossible de demander en résultat, à la fois une colonne et une fonction sur cette colonne. Par exemple : SELECT SALAIRE, SUM(SALAIRE) est invalide.

Un select de groupe peut être utilisé dans une sous interrogation :

Exemple :

Donner le numéro, le nom et le matériau des pièces dont la quantité est la plus élevée.

```
SELECT NP, NOM, MATERIAU
FROM PIECE
WHERE NP IN SELECT NP
                FROM FOURNITURE
                WHERE QTE = SELECT MAX(QTE)
                                FROM FOURNITURE;
```

Calcul sur plusieurs groupes : GROUP BY

Il est possible de subdiviser la table en groupe, chaque groupe étant l'ensemble des lignes ayant une valeur commune. Ceci s'effectue par :

GROUP BY exp1 [,exp2,…]

Qui définit les groupes comme les ensembles de lignes pour lesquelles exp. prend la même valeur.
Un select de groupe avec une clause GROUP BY donnera une ligne résultat pour chaque groupe.

Exemple1 :

Pour chaque pièce fournie, donner le numéro de pièce et le nombre de fournisseurs qui la fournissent.

```
SELECT NP, COUNT(NF)
FROM FOURNITURE
GROUP BY NP
```

Ici le GROUP BY partitionne la table FOURNITURE par NP puis le SELECT donne la valeur unique de pièce et le compte correspondant des fournisseurs pour chaque partition.
En général, quand le GROUP BY est utilisé, chaque item dans le SELECT doit avoir la propriété d'unicité pour tout un groupe et non pas individuellement monovalué pour un groupe.

Exemple2 :

Soit la relation EMPLOYE (NSS, NOM, PRENOM, NPROJ, SAL, NDEP)
On veut calculer le salaire moyen par département et par projet.

```
SELECT NDEP, NPROJ, AVG(SAL)
FROM EMPLOYE
GROUP BY NDEP, NPROJ;
```

Cohérence du résultat :

Dans la liste des colonnes résultats d'un SELECT de groupe ne peuvent figurer que des caractéristiques de groupe, i.e.,

1. des fonctions de groupe
2. des expressions figurant dans le GROUP BY

Sélection des groupes : HAVING

De la même façon qu'il est possible de sélectionner certaines lignes au moyen d'une clause WHERE, il est possible dans un SELECT de groupe de sélectionner certains groupes par la clause HAVING qui se place après la clause GROUP BY.

Syntaxe :

```
SELECT att. [, att., …]
FROM relation [, relation,…]
[WHERE prédicat]
GROUP BY exp. [, exp.,…]
HAVING PREDICAT;
```

Le prédicat figurant dans la clause HAVING suit les mêmes règles de syntaxe qu'un prédicat figurant dans la clause WHERE. Cependant, il ne peut porter que sur des caractéristiques de groupes : fonction de groupe ou expressions figurant dans la clause GROUP BY.

Exemple1 :

Liste des salaires moyens par projet pour les groupes ayant plus de deux employés.

```
SELECT NPROJ, COUNT(*), AVG(SAL)
FROM EMPLOYE
GROUP BY NPROJ
HAVING COUNT(*) > 2 ;
```

Un SELECT de group peut contenir à la fois une CLAUSE WHERE et une clause HAVING. Dans ce cas la clause WHERE doit être placée avant la clause GROUP BY : la clause WHERE sera d'abord appliquée pour sélectionner les lignes puis les groupes seront constitués à partir des lignes sélectionnées et les fonctions de groupes seront évaluées.

Exemple2 :

Donner le numéro de pièces pour toutes les pièces fournies par plus d'un fournisseur.

```
SELECT NP
FROM FOURNITURE
GROUP BY NP
HAVING COUNT(NF) > 1;
```

Une clause HAVING peut donc comporter une sous-interrogation.

Exemple3 :

Quel est le département ayant le plus d'employés

```
SELECT NDEP, COUNT (*)
FROM EMPLOYE
GROUP BY NDEP
HAVING COUNT(*) = ( SELECT MAX(COUNT(*))
                    FROM EMPLOYE
                    GROUP BY NDEP);
```