Chapitre 3 : Algèbre relationnelle

1. Définition de l'algèbre relationnelle

L'algèbre relationnelle a été inventée par E. Codd comme une collection d'opérations formelles qui agissent sur des relations et produisent les relations en résultats [1]. C'est un langage opérationnel, les requêtes sont écrites comme une succession d'opérations effectuées sur des relations. Cet algèbre permet la spécification des opérations à exécuter afin de calculer le résultat d'une requête.

Selon Gardarin [1], ces opérations sont classées selon leurs caractéristiques en :

- Opérations de **base** permettent de déduire les autres. Il existe deux types d'opérations de base ; opérations *ensemblistes* et opérations *spécifiques*.
  Les opérations ensembliste ou binaire englobent : l'union, la différence et le produit cartésien. Alors que les opérations spécifiques ou unaires concernent : la projection, la sélection, la thêta jointure et la jointure naturelle.

- Opérations **dérivées** (obtenues par combinaison des opérations de base). Parmi ses opérations, on a : l'intersection, la jointure externe, la semi-jointure et la division.

2. Opérations de base

2.1. Opérations et opérateurs unaires (spécifiques)

Les opérations spécifiques sont les opérations unaires, cela veut dire que à partir d'une relation, une autre est construite.

2.1.1. Sélection (Restriction)

La restriction (sélection) consiste à créer à partir d'une relation R1(A1, A2,.., An), une relation R2(A1, A2,…,An) de même schéma, mais dont les tuples sont ceux de R1 vérifiant une condition C. Les conditions sont du type :

<Attribut> <Opérateur> <Valeur>

Où l'opérateur appartient à l'ensemble {=, <, ≤, ≥, >, ≠}.

Notations

- $\sigma_{condition}$ (R1)
- R1[Condition]
- RESTRICT(R1, Condition)

Notation graphique

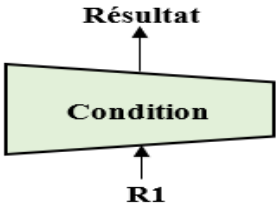

Figure 3.1. Représentation graphique de la sélection.

Exemple : Soit la table Client en extension :

NumC	Nom	Age
101	Ahmed	38
102	Farid	20
103	Ali	25
104	Mohammed	40

$\sigma_{Age>30}$ (Client)

NumC	Nom	Age
101	Ahmed	38
104	Mohammed	40

2.1.2. Projection

La projection d'une relation R1(A1,…,An) sur les attributs (A_i,…,Am), m<n, consiste à créer une relation R2 de schéma Ai,…,Am obtenus en supprimant de la relation R1 les attributs non-mentionnés en opérandes et en éliminant les tuples en double risquant d'apparaître dans la nouvelle table.

Notations

- $\Pi_{A1, A2... Am}$ (R1)
- R1 [Ai, Aj... Am]
- PROJECT (R1, A1, A2,…, Am)

Notation graphique

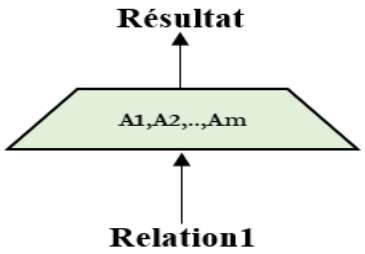

Figure 3.2. Représentation graphique de la projection.

Exemple : Soit la relation R(Num, Nom, Tel)

En extension :

Num	Nom	Tel
101	Ahmed	021352638
102	Farid	021584231

Π_{Nom} (R)

Nom
Ahmed
Farid

2.1.3. Thêta Jointure

La thêta Jointure de deux relations R1 et R2 selon une condition C, est une nouvelle relation R3. Le schéma de R3 est la concaténation des attributs de R1 et R2 et les tuples sont ceux du produit cartésien entre R1 et R2 vérifiant la condition C. La condition C est de la forme :

<Attribut> <Opérateur> <Valeur>

Les opérateurs peuvent être arithmétiques (=,<, ≤, ≥, >, ≠) *ou* logique (Et, Ou, Non).

La thêta-jointure est équivalente à un produit cartésien suivi d'une opération de sélection.

Lorsque l'opérateur est l'égalité « = », on parle alors d'une **Equi-jointure** sinon (i.e. si l'opérateur est l'inégalité « ≠ ») c'est une **Inéqui-jointure**.

Notation

JOIN(R1,R2,Condition)

Représentation graphique

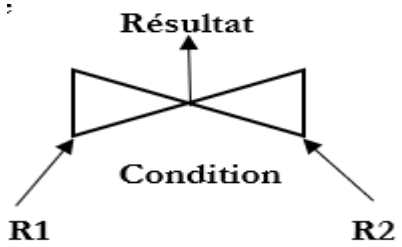

Figure 3.3. Représentation graphique de la jointure.

Exemple :

Soit les deux relations : **Employé**(NomE, SalaireE) et **Chef**(NomC, SalaireC).

Employé

NomE	SalaireE
Taleb	20000
Mansouri	10000
Benmohammed	6000

Chef

NomC	SalaireC
Dali	25000
Tir	12000

Employé $\bowtie_{\text{SlaireE>SalaireC}}$ **Chef**

NomE	SalaireE	NomC	SalaireC
Taleb	20000	Tir	12000

Cette opération permet de répondre à la question : quels sont les noms des employés qui gagnent plus que le chef.

2.1.4. Jointure naturelle

La jointure naturelle entre deux relations R1 et R2 qui n'ont pas forcément le même schéma, forme une troisième relation R3 dont les attributs sont obtenus par concaténation des attributs de R1 et R2. Les tuples de R3 sont ceux de R1 et de R2 ayant mêmes valeurs pour les attributs de même nom (respectant une équijointure entre les attribut communs). Ces attributs communs ne sont pas dupliqués dans la relation R3, mais fusionnés en une seule colonne.

Notation

JOIN(R1,R2)

Représentation graphique

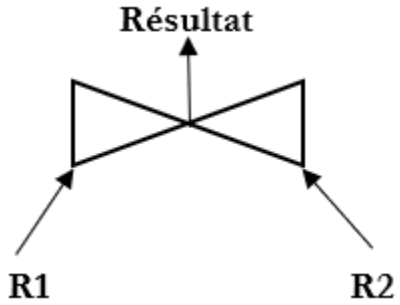

Figure 3.4. Représentation graphique de la jointure naturelle.

Exemple :

Ami

Nom	Prénom	Age
Taleb	Amira	6
Mansouri	Ali	40
Benmohammed	Farid	20
Merouani	Mouna	6

Cadeau

Age	Article	Prix
40	Parfum	120
6	Montre	320
20	Garniture	82

R = Ami ⋈ Cadeau

Nom	Prénom	Age	Article	Prix
Taleb	Amira	6	Montre	320
Mansouri	Ali	40	Parfum	120
Benmohammed	Farid	20	Garniture	82
Merouani	Mouna	6	Montre	320

2.1.5. Traduction en SQL

2.1.5.1. Requêtes simples (SELECT-FROM)

L'opérateur de projection $\Pi_{(A1,\ldots An)}$(relation) se traduit en SQL par la requête :

SELECT A_1, ..., A_n FROM relation

- SELECT : Indique la liste des attributs qui constituent le résultat.
- FROM : Indique la (ou les) tables dans lesquelles se trouve les attributs utiles à la requête.
- Lorsqu'on ajoute DISTINCT après SELECT cela permet de demander explicitement l'élimination des doubles car SQL n'élimine pas les doubles, tandis que ALL est l'option par défaut.

2.1.5.2. Sélection de colonne (clause WHERE)

L'opérateur de sélection $\sigma_{(condition)}$(relation) se traduit en SQL par la requête :

SELECT * **FROM** relation **WHERE** condition

WHERE permet de spécifier les critères de sélection. Cette clause indique les conditions que doivent satisfaire les n-uplets pour faire partie du résultat. En SQL ces critères sont très riches, dans ce cours nous nous contentons de présenter les plus utilisés. Notons que ***** veut dire toutes les colonnes.

Notons que l'omission de la clause WHERE nous permet d'afficher l'intégralité d'une table, et avoir ainsi toutes les lignes et toutes les colonnes, on peut lister tous les attributs *ou* utiliser le caractère * ayant la même signification.

Conditions possibles dans la clause WHERE :

Condition de base de **comparaison** à l'aide des opérateurs {=, ≠, <, >, <=, >=} :

Colonne = | < | > | <> | <= | >= **Constante**

Exemple :

SELECT *

FROM Pays

WHERE population < 20 ;

Condition **BETWEEN** permettant de tester l'appartenance à un intervalle (i.e. obtenir une recherche par intervalle). Notons que NOT BETWEEN est utilisé pour la négation.

- **WHERE** population >= 50 **AND** population <= 60

Équivalent à :

- **WHERE** population **BETWEEN** 50 **AND** 60

Condition **LIKE** de comparaison de chaînes de caractères.

Colonne [NOT] LIKE modèle de chaîne

Le modèle de chaîne peut contenir n'importe quel caractère et les caractères 'chaîne' contient des caractères jokers (%, _)

Exemples :

Quel sont les produits dont le nom commence par 'P'.

SELECT *

FROM Produit

WHERE Desig **Like** "P%"

Autres exemples :

- WHERE pays LIKE '%lande' -> Irlande, Islande, Finlande, Hollande
- WHERE pays LIKE '%ran%' -> Iran, France
- WHERE pays LIKE 'I_lande' -> Irlande, Islande

Condition **NULL** de test de nullité permettant de vérifier si une colonne est nulle ou non, car en pratique il est possible d'avoir des valeurs non définies (unknown).

Colonne IS [NOT] NULL

Exemple : Les commandes avec une quantité différente de NULL.

SELECT *

FROM COMMANDE

WHERE QTE IS **NOT NULL** ;

Condition **IN** permettant de tester l'appartenance à un ensemble de valeurs possibles. Cet ensemble peut être spécifié explicitement ou peut être le résultat de *sous-requêtes* (i.e. requêtes imbriquées). Notons que NOT IN indique la non appartenance à un ensemble.

Exemple 1 :

WHERE monnaie = 'Euro' **OR** monnaie = 'Dollar' **OR** monnaie = 'Dinar'

Équivalent à :

WHERE monnaie **IN** ('Euro', 'Dollar', 'Dinar')

La condition **EXISTS** (sous-requête) renvoi vrai quand le résultat de la sous-requête n'est pas vide. Le contraire **NOT EXISTS** (sous-requête).

Exemple : Soient les deux relations :

Fournisseur (FournisseurID, FournisseurNom, ContactNom, Address)

Produit (ProduitID, NomP, FournisseurID, CategoriID, Unite, Prix)

Donner les fournisseurs ayant fourni des produits dont le prix est inférieur à 20.

```
SELECT FournisseurNom
FROM Fournisseur
WHERE EXISTS (SELECT NomP FROM Produit WHERE Produit.FournissurID=
Fournissur.FournissurID AND Prix < 20);
```

2.1.5.3. Tri de résultats (ORDER BY)

La clause **ORDER BY** permet de trier les n-uplets du résultat d'une requête. Une liste d'attributs servant de critère au tri doit suivre cette clause qui a la syntaxe suivante :

ORDER BY colonne [**DESC**]

Afin de trier en ordre descendant, on ajoute **DESC** après la liste des attributs et pour avoir un tri ascendant, on ajoute **ASC**. Si l'ordre de tri n'est pas spécifié, par défaut les attributs seront triés en ordre ascendant.

Exemple : Donner la liste des employés du département 10 par ordre décroissant de leur salaire.

SELECT Num_Dept, Nom_Emp, Salaire

FROM EMPLOYES

WHERE Num_Dept = 10 **ORDER BY** Salaire **DESC**;

Les opérations de thêta jointure et de la jointure naturelle seront traduites dans la section 5.5.

2.2. Opérations ensemblistes (binaires)

Les opérations ensemblistes sont des opérations binaires (i.e. à partir de deux relations, une troisième relation peut être construite).

2.2.1. Union

L'union de deux relations R1 et R2 ayant le même schéma, est une nouvelle relation R. Les tuples de la nouvelle relation R appartiennent à : R1 ou R2 *ou bien* R1 et R2.

Notation

- R= R1 ⋃ R2
- *UNION(R1,R2)*
- *APPEND (RELATION1, RELATION2)*

Modélisation graphique

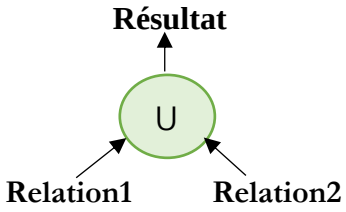

Figure 3.5. Représentation graphique de l'union.

Exemple : Soit les deux relations Acteur1 et Acteur2 dont l'extension est comme suit.

Acteur1

Nom	Prénom	Datenaissance
DiCaprio	Leonardo	11/11/1974
Spacey	Kevin	26/ 07/1959
Eastwood	Clint	31 /05/ 1930
Freeman	Morgan	1/06/ 1937

Acteur2

Nom	Prénom	Datenaissance
Freeman	Morgan	1/06/ 1937
Johnny	Depp	9 /06/ 1963

Acteur1 ⋃Acteur2= UnionActeur

Nom	Prénom	Datenaissance
DiCaprio	Leonardo	11/11/1974
Spacey	Kevin	26/ 07/1959
Eastwood	Clint	31 /05/ 1930
Freeman	Morgan	1/06/ 1937
Johnny	Depp	9 /06/ 1963

2.2.2. Différence

La différence est une opération qui porte sur deux relations R1 et R2 de même schéma servent à construire une nouvelle relation R3 de même schéma. Les tuples de la relation résultante R3 appartiennent à R1 et n'appartiennent pas à R2. Il est à noter que l'ordre des relations opérandes est important car la différence est un opérateur qui n'est pas commutatif (i.e. $R2-R1 \neq R1-R2$).

Notation

- R3= R1 – R2
- DIFFERENCE (R1, R2)
- REMOVE (R1, R2)
- MINUS (R1, R2)

Représentation graphique

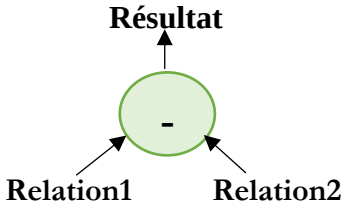

Figure 3.6. Représentation graphique de la différence.

Exemple :

Client1

NumCl	NomCl	Tel
101	Ahmed	032482564
102	Mohammed	021256687

Client2

NumCl	NomCl	Tel
102	Mohammed	021256687

Client1-Client2=DifClient

NumCl	NomCl	Tel
101	Ahmed	032482564

2.2.3. Produit cartésien (Cartesian product)

Le produit cartésien de deux relations R1 et R2 qui n'ont pas obligatoirement le même schéma, est une relation R3. Le schéma de la relation R3 est constitué de la concaténation des attributs du schéma de R1 et de R2 et dont les tuples sont constitués de toutes les combinaisons possibles entre les tuples de R1 et les tuples de R2.

Notation

- R1 X R2
- PRODUCT (R1, R2)
- TIMES (R1, R2)

Représentation graphique

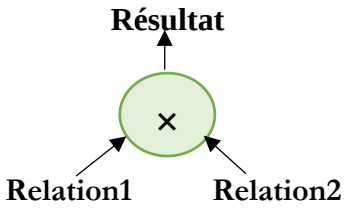

Figure 3.7. Représentation graphique du produit cartésien.

Exemple : Soit la relation : Pays (Nom, Capitale, Monnaie).

En extension :

Pays

Nom	Capitale	Monnaie
Algérie	Alger	1
Maroc	Rabat	6

Soit la relation : Monnaie (Num, Nom).

En extension :

Monnaie

Num	Nom
1	Dinar Algérien
2	Euro
6	Dirham

Pays× Monnaie= ProduiPM

P.Nom	Capitale	Monnaie	Num	M.nom
Algérie	Alger	1	1	Dinar Algérien
Maroc	Rabat	6	1	Dinar Algérien
Algérie	Alger	1	2	Euro
Maroc	Rabat	6	2	Euro
Algérie	Alger	1	6	Dirham
Maroc	Rabat	6	6	Dirham

Remarque : Si les deux relations R1 et R2 ont un attribut de même nom, on renomme cet attribut ou bien en spécifie le nom de la relation à laquelle il appartient (e.g., R1.A, R2.A).

3. Opérations dérivées

3.1. Intersection

L'intersection de deux relations de même schéma R1 et R2 est une nouvelle relation R3 de même schéma ayant pour tuples ceux appartenant à R1 et à R2.

Notation

- R1 ∩ R2
- INTERSECT (R1, R2)
- AND (R1, R2)

Modélisation graphique

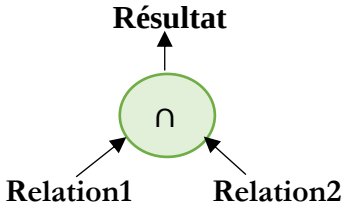

Figure 3.8. Représentation graphique de l'intersection.

Exemple : Considérant l'exemple suivant *tiré de [2]*.

Deux bibliothèques B1 et B2 fusionnent et décident de rechercher les livres qu'elles ont en commun pour n'en garder un seul exemplaire. Chaque bibliothèque possède une relation LivreB1 et LivreB2.

LivreB1

NLIV	TITRE	NOMAUT
101	Les fourmis	Bertrand
102	Le soir des fourmis	Bertrand
210	La révolte des fourmis	Bertrand
104	Les 10 mousquetaires	Artagnan

LivreB2

NLIV	TITRE	NOMAUT
101	La basilique de Paris	Victorien
102	Le soir des fourmis	Bertrand
210	Le roi de la forêt	Aiglon
104	Les 10 mousquetaires	Artagnan

LivreB2 ∩ LivreB2= INTERLIVRE

NLIV	TITRE	NOMAUT
102	Le soir des fourmis	Bertrand
104	Les 10 mousquetaires	Artagnan

3.2. Jointure externe

La jointure externe entre deux relations R1 et R2 de schéma quelconque est une relation R3 dont le schéma est la concaténation des attributs de R1 et de ceux de R2 en ne représentant les attributs ayant

le même nom qu'une seule fois. Les tuples de R3 sont ceux obtenus avec une jointure naturelle entre R1 et R2 et ceux de R 1 et de R2 ne participants pas à la jointure en représentant par des valeurs nulles ceux de l'autre relation.

Dans le but de conserver les tuples quand les relations jointes n'ont pas de projection identique sur l'attribut de jointure, Codd a introduit les jointures externes qui sont nécessaires en particulier pour composer des vues sans perte d'informations.

La jointure externe entre deux relations R1 et R2 génère une relation R3 qui a comme :

- Attributs la concaténation des attributs de R1 et de R2 en ne représentant les attributs commun qu'une seule fois.
- Tuples ceux obtenus avec une jointure naturelle entre R1 et R2 et ceux des deux relations initiales R 1 et R2 ne participants pas à la jointure en représentant par des valeurs nulles ceux de l'autre relation.

Notation

EXT-JOIN(R1,R2)

Modélisation graphique

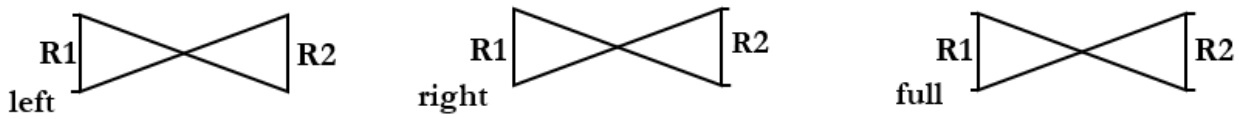

Figure 3.9. Représentation graphique de la jointure externe.

Notons que pour la jointure externe gauche notée LEXT-JOIN, on garde tous les n-uplets de la première table (gauche). Tandis que pour la jointure externe droite notée REXT-JOIN, on garde tous les n-uplets de la deuxième table (droite).

Exemple **:** prenant l'exemple de deux tables Client et Commande dont l'attribut commun est le numéro du client.

Client

NumClient	NomClient	PrénomClient	NumTel
1001	Mansouri	Ali	032482223
1002	Ahmed	Salem	032485760
1003	Mohammed	Talbi	032488080
1004	Yacine	Hafsa	032486666

Commande

NumCommande	NumClient	Total
C001	1001	1800,00
C002	1003	2364,76
C003	1001	3000,00

Jointure externe de Client et Commande

NumClient	NomClient	PrénomClient	NumTel	NumCommande	Total
1001	Mansouri	Ali	032482223	C001	1800,00
1002	Ahmed	Salem	032485760	-	-
1003	Mohammed	Talbi	032488080	C002	2364,76
1001	Mansouri	Ali	032482223	C003	3000,00
1004	Yacine	Hafsa	032486666	-	-

Comme les clients 1002 et 1004 n'ont pas de commandes dans la table Commande, les attributs NumCommande et Total prennent la valeur NULLE. Cette opération a permis de garder les clients sans commande. Donc, elle est utile en pratique.

3.3. Semi jointure

La semi-jointure de deux relationsR1 et R2 est une relation R3 ayant le schéma de R1 et les tuples sont ceux de R1 participant à la jointure naturelle entre R1 et R2. Ici encore, R1et R2 sont de schéma de quelconque.

Notation

- **R1 ⋉ R2**
- **SEMI-JOIN (R1, R2)**

Présentation graphique

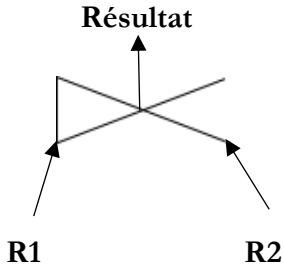

Figure 3.10. Modélisation graphique de la semi jointure.

Exemple :

Client

NumClient	NomClient	PrénomClient	NumTel
1001	Mansouri	Ali	032482223
1002	Ahmed	Salem	032485760
1003	Mohammed	Talbi	032488080
1004	Yacine	Hafsa	032486666

Commande

NumCommande	NumClient	Total
C001	1001	1800,00
C002	1003	2364,76

Client ⋉ **Commande**

NumClient	NomClient	PrénomClient	NumTel
1001	Mansouri	Ali	032482223
1003	Mohammed	Talbi	032488080

Note : Lorsqu'il n'est pas nécessaire de conserver tous les attributs des deux relations (i.e. seuls les attributs d'une des deux relations sont conservés en résultat), la semi-jointure est très utile afin d'optimiser l'évaluation des questions.

3.4. Division

C'est une opération portant sur deux relations R1 et R2, telles que le schéma de R2 est inclus strictement dans celui de R1. La relation *générée R3* aura le schéma de R1 moins les attributs de R2 et est formée de tous les tuples qui concaténés à chaque tuple de R2 donnent toujours un tuple de R1. La relation R_2 ne peut pas être vide. Si R_1 est vide, la relation résultante de la division est vide.

L'intérêt de cette opération réside dans le fait qu'elle permet de rechercher dans une table les sous-tables qui sont complétées par tous ceux d'une autre table et de répondre à des requêtes de la forme : « quel que soit X, trouver Y ».

Notation

R1/R2

DIVISION(R1,R2)

Modélisation graphique

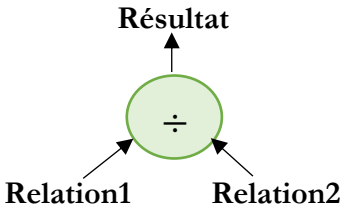

Figure 3.11. Modélisation graphique de la division.

Exemple :

Pièce

Nom	Couleur	Poids	Ville
Clou	noir	20	Constantine
Clou	bleu	25	Alger
Vis	noir	20	Constantine
Boulon	rouge	23	Oran
Vis	bleu	25	Alger
Boulon	vert	18	Constantine

Nature

Nom
Vis
Clou

Pièce ÷ Nature= PNature

Couleur	Poids	Ville
Noir	20	Constantine
Bleu	25	Alger

4. Le langage algébrique

Le langage algébrique est un langage d'interrogation de BDD. Les opérations de base de l'algèbre relationnelle constituent un langage complet. Une expression algébrique peut être représentée sous forme d'arbre dont : la racine correspond à la requête, les nœuds correspondent aux opérateurs algébriques et les feuilles correspondent aux relations. L'objectif est de mieux comprendre l'optimisation logique de requête.

Exemple : *tiré des sources de la société Oracle.*

EMP(**ENO**, ENOM, PROF, DATEEMB, SAL, COMM, DNO).

DEPT(**DNO,** DNOM, DIR, VILLE).

Donner les numéros des employés travaillant à Boston.

$R = \Pi_{ENO}(EMP \bowtie \sigma_{VILLE='Boston'}(DEPT))$.

L'arbre algébrique est :

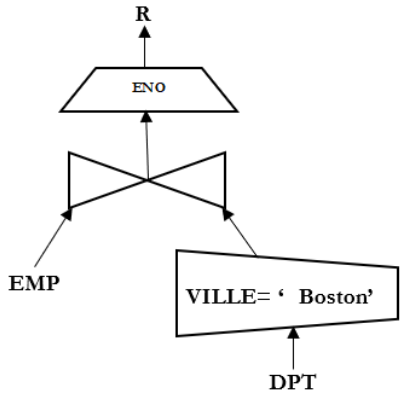

Figure 3.12. Arbre algébrique correspondant à l'exemple.

5. Traduction en SQL

Cette section présentera la traduction en SQL des opérations ensemblistes (binaires) et des opérations dérivées.

5.1. Union

L'opérateur d'union permet de *fusionner* deux tables ayant le même schéma et se traduit tout simplement en SQL par la requête :

```
SELECT * FROM relation_1 UNION SELECT * FROM relation_2
```

5.2. Intersection

L'opérateur d'intersection permet d'obtenir les lignes *communes* de deux tables et se traduit en SQL comme suit :

```
SELECT * FROM relation_1 INTERSECT SELECT * FROM relation_2
```

Exemple : Soient les relations

Etudiant (NumEtud, ENom, DateN)

Cours (NumCour, CNom, Ens)

Examen (NumEtud, NumCour, Note)

Donner les numéros des étudiants qui ont soutenu l'examen de numéro cours 02 et aussi l'examen de numéro cours03.

```
SELECT NumEtud FROM Examen WHERE NumCour = 02

INTERSECT

SELECT NumEtud FROM Examen WHER NumCour = 03 ;
```

5.3. Différence

L'opérateur MINUS permet d'effectuer la différence entre deux tables et se traduit SQL par la requête suivante :

```
SELECT * FROM relation_1 EXCEPT SELECT * FROM relation_2
```

Exemple :

Donner les numéros des étudiants qui ont soutenu l'examen de Numéro Cours 02 mais qui n'ont pas soutenu l'examen de Numéro Cours 03.

```
(SELECT NumEtud FROM Examen WHERE NumCour = 02)
EXCEPT
(SELECT NumEtud FROM Examen WHERE NumCour = 03) ;
```

5.4. Produit cartésien (sans jointure)

Le produit cartésien $Relation_1 \times Relation_2$ se traduit en SQL par une simple requête entre deux tables comme suit :

```
SELECT * FROM relation_1, relation_2
```

Il peut aussi s'écrire en utilisant le mot-clé JOIN destiné aux jointures comme suit :

```
SELECT * FROM table_1 CROSS JOIN table_2
```

Remarque : SQL ne comporte pas d'opérateurs spécifiques permettant de réaliser directement une division. Néanmoins, il est possible d'exprimer la sémantique de la division en utilisant les opérateurs logiques.

5.5. Jointure de tables (condition de jointure)

La requête SELECT avec une seule relation dans la clause FROM permet de réaliser des sélections et des projections. Par contre, avec plusieurs relations dans la clause FROM réalise une jointure (et produit cartésiens). Afin d'exprimer des jointures avec SQL, il existe plusieurs manières dont la syntaxe globale possible est la suivante :

table_1 [INNER | { { LEFT | RIGHT | FULL } [OUTER] }] JOIN table_2 ON predicat [...]

table_1 [INNER | { { LEFT | RIGHT | FULL } [OUTER] }] JOIN table_2 USING (colonnes) [...]

table_1 NATURAL [INNER | { { LEFT | RIGHT | FULL } [OUTER] }] JOIN table_2 [...]

5.5.1. Thêta jointure

La thêta-jointure est exprimée avec une condition sur les colonnes communes des deux tables dans la clause Where en utilisant une requête simple. La thêta-jointure utilise n'importe quel opérateur de comparaison pour joindre deux tables.

SELECT att1, att2,...
FROM table1 INNERJOIN table2
ON table1.attx θ table2.attx
[WHERE autres_conditions];

- θ est un opérateur parmi =,<, ≤, ≥, >, ≠, like, ...
- La condition de jointure table1.attx θ table2.attx est exprimée avec le mot **ON.**
- Les autres conditions sont appliquées après la condition de jointure.
- Lorsque la thêta-jointure utilise seulement la condition d'égalité, elle devient une équi-jointure.

Ou plus simplement :

SELECT ...
FROM <table gauche>
[INNER]JOIN <table droite>
ON <condition de jointure>

Exemple 1 :

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT
    INNER JOIN T_TELEPHONE
        ON T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID
```

Ou en utilisant le sur nommage :

```
SELECT CLI_NOM, TEL_NUMERO
FROM   T_CLIENT C
      INNER JOIN T_TELEPHONE T
          ON C.CLI_ID = T.CLI_ID
```

Exemple 2 :

```
SELECT * FROM film, realisateur WHERE film.id_real = realisateur.id_real;
SELECT * FROM film JOIN realisateur ON film.id_real = realisateur.id_real;
SELECT * FROM film INNER JOIN realisateur ON film.id_real = realisateur.id_real;
```

Exemple 3 :

Les numéros des fournisseurs et des clients qui sont situés dans la même ville. L'attribut ville est commun entre les deux relations.

```
SELECT NumF, NumC
  FROM FOURNISSEUR, CLIENT
 WHERE FOURNISSEUR.VILLE = CLIENT.VILLE;
```

5.5.2. Jointure naturelle

La jointure naturelle peut être exprimée en utilisant la clause NATURAL JOIN comme suit :

```
SELECT ...
FROM   <table gauche>
      NATURAL JOIN <table droite>
            [USING <noms de colonnes>]
```

Exemple : *Exemple tiré de [3].*

Soient les deux relations : Realisateur(id_real, nom, prenom) et Film(id_film, id_real, titre).
La jointure naturelle entre la table film et la table realisateur est comme suit:

```
SELECT * FROM film NATURAL JOIN realisateur;
```

Il est possible aussi d'exprimer la jointure naturelle avec une thêta-jointure en mentionnant dans la clause USING les colonnes en commun comme suit :

```
SELECT * FROM film JOIN realisateur USING (id_real);
SELECT * FROM film INNER JOIN realisateur USING (id_real);
```

5.5.3. Jointure externe

Les jointures externes à gauche, à droite et globale sont obtenues avec la clause: left | right | full outer join ...ON condition. La syntaxe est comme suit :

SELECT ...
FROM <table gauche>
LEFT | RIGHT | FULL OUTER JOIN <table droite>
ON condition de jointure

Note: Les mots clé **OUTER** et **INNER** sont facultatifs.

Exemple : Donner les avions, y compris ceux qui ne sont affectés à un vol.

SELECT Num
FROM Avion LEFT OUTER JOIN Vol
ON Avion.Num=Vol.Num;

6. Fonctions d'agrégat

Les fonctions d'agrégat sont un ensemble de fonctions qui permettent d'effectuer des statistiques sur le résultat d'une requête [8] (MIN : minimum, MAX : maximum, SUM : somme, AVG : moyenne, COUNT : nombre, etc.).

- 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 calculer la valeur maximale d'un ensemble ;
- MIN permet de calculer la valeur minimale d'un ensemble.

Exemples :

Soit la base de données suivante qu'on va utiliser pour exemplifier les fonctions d'agrégat :

Emp (Eno, Ename, Title, City).

Project (Pno, Pname, Budget, City).

Pay(Title, Salary).

Works (Eno, Pno, Resp, Dur).

Q1. Quel est le nombre des employés Constantinois ?

```
SELECT COUNT(*)
  FROM Emp
WHERE City='Constantine';
```

Q2. Quel est le plus grand salaire et le plus petit salaire de toutes les professions ?

```
SELECT MAX(Salary), MIN(Salary)
  FROM Pay;
```

Q3. Quels sont les noms des professions qui payent le plus et les salaires correspondant) ?

```
SELECT Title, Salary
  FROM Pay
WHERE Salary = (SELECT MAX(Salary)
                FROM Pay);
```

Q4. Donner les budgets totaux des budgets des projets d'Alger ?

```
SELECT SUM(Budget)
  FROM Project
WHERE City = 'Alger';
```

Q4. Quel est le nombre de villes où il y a un projet avec l'employé E101?

```
SELECT COUNT (DISTINCT City)
  FROM Project, Works
WHERE Project.Pno = Works.Pno AND Works.Eno = 'E101';
```

Q 5. Quels sont les noms des projets dont le budget est supérieur au budget moyen?

```
SELECT Pname
  FROM Project
WHERE Budget > (SELECT AVG(Budget)
                FROM Project);
```

7. Clause GROUP BY …HAVING

7.1. La clause GROUP BY

La clause **GROUP BY** permet de définir des groupes pour partitionner les n-uplets résultats en fonction des valeurs de certains attributs en groupes homogènes.

Cette clause est utile lorsqu'on veut effectuer des calculs sur des *groupes* de lignes (e.g., le salaire moyen le plus bas par profession, etc.). Notons que la clause GROUP BY doit être utilisée lorsqu'en utilise des calculs au niveau de la clause SELECT sinon, le résultat n'aura pas un sens.

Exemple : Donner le salaire moyen le plus bas par profession.

```
SELECT MIN(AVG(SAL))
   FROM EMP
 GROUP BY PROF ;
```

7.2. La clause HAVING

La clause **HAVING** spécifie un filtre (i.e. condition de regroupement des n-uplets) sur les résultats en exprimant des conditions sur le groupe de lignes générés. Le but est de ne sélectionner que certains groupes. Cette clause se place après la clause GROUP BY.

Une requête de groupement qui comporte une clause GROUP BY, peut contenir une clause WHERE et une clause HAVING. Dans ce cas : D'abord, la clause WHERE est appliquée afin de sélectionner les lignes. Puis, les groupes seront constitués à partir des lignes sélectionnées, ensuite, les fonctions de groupe seront évaluées. Enfin, et la clause HAVING sera enfin appliquée pour sélectionner les groupes. La condition ne porte pas sur chaque tuple de la table comme pour le where mais sur l'ensemble des tuples d'un groupe.

SELECT ... FROM ... WHERE ... GROUP BY ... HAVING <condition>

Exemple 1 : Soit la table "Achat" qui contient les achats de différents clients avec le coût du panier pour chaque achat.

id	Client	tarif	date_achat
1	Ahmed	102	2018-10-23
2	Ali	47	2018-10-27
3	Asma	18	2018-11-05
4	Asma	20	2018-11-14
5	Ahmed	160	2018-12-03

Q1 . Donner la liste des clients qui ont commandé plus de 40 Dinar, toute commande confondue.

SELECT client, **SUM**(tarif)
FROM achat
GROUP BY client
HAVING SUM(tarif) > 40;

Résultat :

client	SUM(tarif)
Ahmed	262
Ali	47

Exemple 2 :

Soit la relation : Pays(Nom, Capital, Population, Surface, Continent).

Q 2. Donner la population totale par continent, tel que chaque continent a une population > 50.

SELECT continent, **SUM**(population)
FROM PAYS **GROUP BY** continent
HAVING SUM(pop) > 50 ;