Chapitre 1 : Les objectifs des Bases de données

Une base de données

• Ensemble de données utilisées par des applications de certaines entreprises

Ex : Une banque, un hôpital, une université, une entreprise de fabrication

Un Système de Gestion de Base de Données (SGBD)

• Ensemble de programmes qui permet de gérer une base de données partagée par plusieurs utilisateurs simultanément :

- ✓ accéder à des données,
- ✓ ajouter des données,
- ✓ mettre à jour des données,
- ✓ supprimer des données

Pourquoi une base de données et un SGBD?

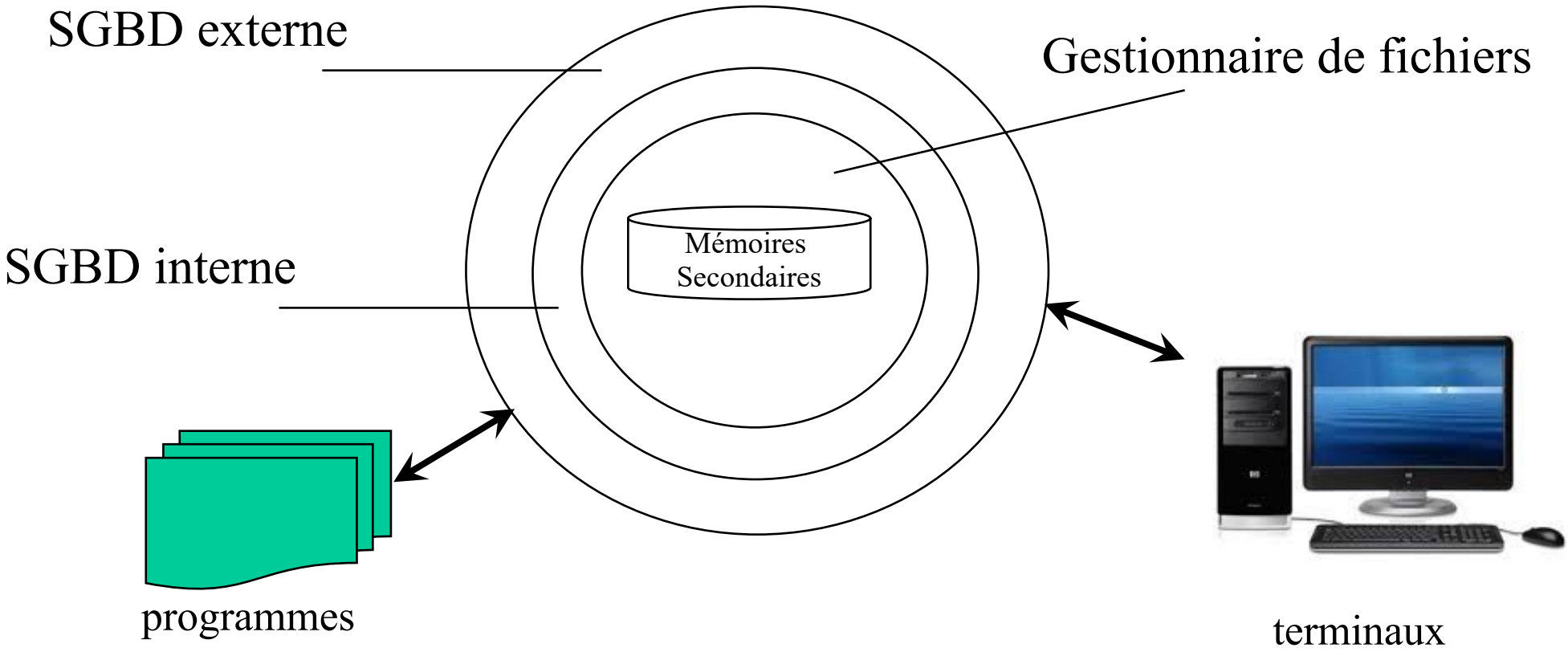
- *Compacité* : nécessité de stocker les données sur mémoire secondaire **Une BD = plusieurs téraoctets (10^{12} octets) …**

- *Rapidité* : accès rapides aux informations demandées par les utilisateurs

- *Maintenance facilitée :* les données sont stockées à un seul endroit

- *Exactitude* : des informations précises et réactualisées sont disponibles à tout moment sur demande.

Vision simplifiée d'un SGBD



Description du contenu d'une BD

– Faite à l'aide d'un *modèle* de données

– Modèle : concepts de structuration + règles d'utilisation

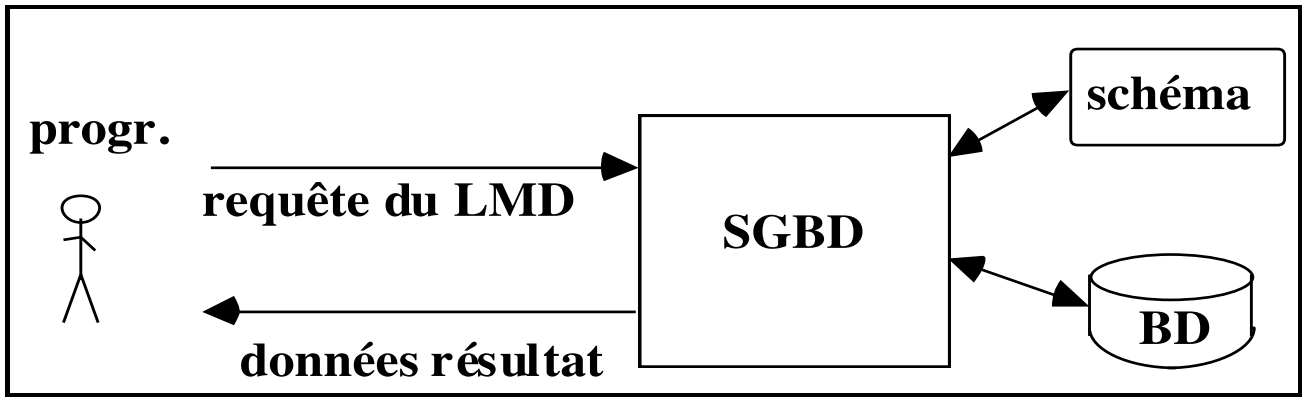
différents modèles : Hiérarchique, Réseaux, Relationnel, Objets, …

– Un SGBD est caractérisé par le modèle de données qu'il utilise

==> création du schéma d'une base de données (description de la structure des données) exprimé à l'aide d'un langage de définition de données (LDD)

Manipulation des données

- avec **un langage de manipulation des données (LMD)**



• Le LMD dépend du modèle de données du SGBD

Avec le modèle relationnel, le LMD est déclaratif : dire QUOI sans dire COMMENT

Ex : quels sont les noms des étudiants inscrits en BUT1 ?

• Le LMD peut être utilisé en interactif ou inclus dans un langage de programmation (JAVA, C, …)

Chapitre 2 : Le modèle Relationnel

- **Inventé par E. Codd en 1970**

UNE RELATION = UN **TABLEAU** A DEUX DIMENSIONS

UNE **COLONNE** = UN ATTRIBUT

EN-TÊTE DU TABLEAU = **SCHEMA** DE LA RELATION
(DESCRIPTION DU TYPE)

UNE **LIGNE** = UN TUPLE

ENSEMBLE DES LIGNES = CONTENU DE LA RELATION
(EXTENSION)

Le modèle Relationnel : 2.1 Présentation informelle

Exemple : la relation Fournisseur

Numéro Fournisseur	Nom Fournisseur	Ville
1435	Helfer	Paris
678	Giard	Nantes
1234	Rolland	Marseille
007	Bond	Londres

Le modèle Relationnel : 2.2 Approche formelle

Domaine : Ensemble de valeurs caractérisé par un nom

Ex:

Produit cartésien d 'un ensemble de domaines D1,D2,… Dn **noté D1 x D2 x … Dn**
= ensemble des éléments (v_1, v_2 …, v_n)
avec $v_1 \in D_1$, $v_2 \in D_2$, … , $v_n \in D_n$

Renault	*blanche*
Renault	*grise*
Peugeot	*blanche*
Peugeot	*grise*
Citroen	*blanche*
Citroen	*grise*

Le modèle Relationnel : 2.2 Approche formelle

- **Relation**

*Soient D1, D2,..., Dn une liste de domaines****, une relation est un sous-ensemble du produit cartésien D1 x D2 x ... x Dn.***

- **Tuple** ***(N-Uplet): un élément d 'une relation.*** $(v_1, v_2 \ldots, v_n)$

Un tuple est unique dans une relation
L'ordre des tuples dans une relation n'a pas d'importance

Cardinalité *d 'une relation* ***= nombre de tuples***

- **Attribut** ***: nom donné au rôle joué par un domaine dans une relation***

Un attribut est unique par relation
L'ordre des attributs n'a pas d'importance.

Degré *d 'une relation* ***= nombre d 'attributs***

Le modèle Relationnel : 2.2 Approche formelle

Schéma d'une Relation : ***nom de la relation suivi par la liste des attributs qui la composent et par la définition de leurs domaines.***

$$R(A1 : D1,..., An : Dn)$$

où R est le nom de la relation, Ai les attributs et Di les domaines associés.

Exemple : le schéma de la relation EMPLOYE est le suivant :

EMPLOYE(NoEmpl : Entiers, Nom : Caractères, Année : Entiers, NumDept : Entiers)

Remarque : Les domaines peuvent être omis

Le schéma définit la relation en intention

La table définit la relation en extension

Le modèle Relationnel : 2.3 Règles d'intégrité structurelle

1. Clé

La clé d'une relation est un ensemble minimal d'attributs dont chaque valeur détermine un tuple unique dans toute extension de la relation.

Exemple

l'attribut NoEmpl peut être la clé de la relation EMPLOYE puisque tout employé possède un numéro unique et que deux employés ne peuvent pas avoir le même numéro

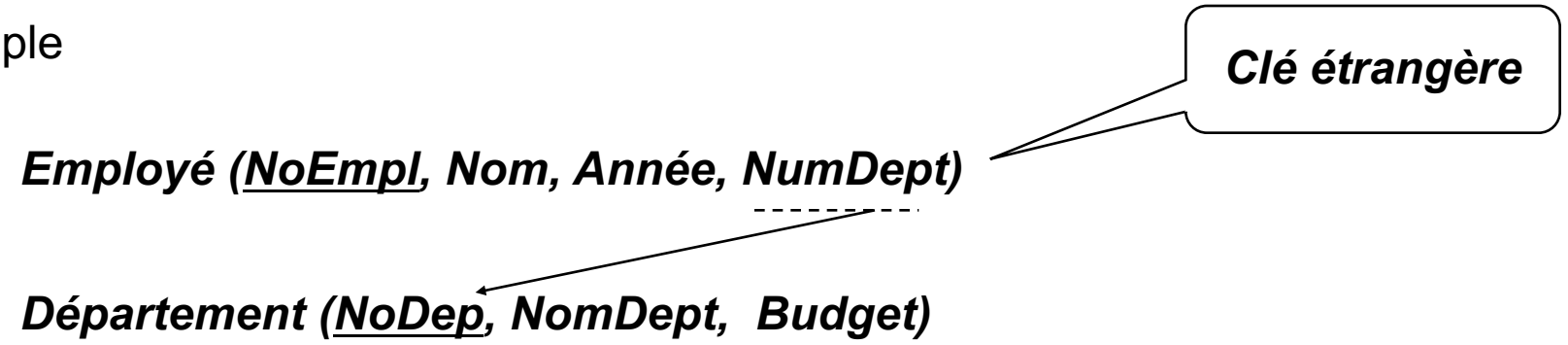
EMPLOYE (NoEmpl, Nom, Année, NumDept)

Le modèle Relationnel : 2.3 Règles d ’intégrité structurelle

2. Contrainte référentielle

Une référence (ou clé étrangère) est un attribut (ou un groupe d ’attributs) dont les valeurs sont celles d ’une clé d ’une autre relation

Exemple



Le modèle Relationnel : 2.3 Règles d ’intégrité structurelle

Toute valeur de l’attribut Numdept de la relation Employé doit se trouver dans l'attribut NoDep de la relation Département

Relation Employé

NoEmpl	Nom	Année	NumDept
1045	Dupond	1988	03
2067	Dupont	1998	09
0456	Martin	2001	10

Relation Département

NoDep	Nomdept	Budget
03	Finances	200 000
09	Commercial	400 000
10	Informatique	260 000
101	Production	560 000

Le modèle Relationnel : 2.3 Règles d ’intégrité structurelle

3. Valeur nulle et clé

La valeur nulle (NULL) est une valeur conventionnellement introduite dans une relation pour représenter une information inconnue ou inapplicable.

Ex: Employé (NoEmpl, Nom, NomMarital, Année, Adresse, Téléphone, Nodept)

Le téléphone non connu à une certaine date
(10, Durand, Faure, 1980, Paris, NULL, 75)

ou
le nom marital pour un employé homme
(23, Fergio, NULL, 1987, Marseille, 0234565412, 13)

Contrainte de relation : Toute relation doit posséder au moins une clé non nulle

Schéma d'une base de données :
schéma des relations et contraintes d'intégrité structurelle

Présentation de SQL

•SQL (Structured Query Language)

• Langage déclaratif permettant :
  • interrogation des données (requêtes)
  • manipulation des données (insertion, suppression, mise à jour)
  • définition des données et des contraintes sur celles-ci
  • définition des vues et d'index

• Différentes normes ISO : SQL-86; SQL-89; SQL-92; SQL-99

Définition des schémas

Le Langage de Définition de Donnés (LDD) permet la définition de la base de données.

Création de relations :

```
CREATE TABLE <nom_relation>
( <élément de relation> < , élément de relation>*
  < , contrainte de relation>*
);
```

<élément de relation> ::= <nom_attribut> <type de données>
[<contrainte d 'attribut>*]

<type de données> ::= VARCHAR <longueur> | INT | REAL | DATE …

- **Chaque relation est définie par un *nom de relation* et une liste d 'attributs**
- **Chaque attribut est défini par un *nom d 'attribut* et un *type de données***

Le modèle Relationnel : 2.4 Définition des schémas de relations

Une contrainte d 'attribut **concerne un seul attribut**

ex : valeur NULL impossible : NOT NULL
Attribut clé : PRIMARY KEY
Unicité de l 'attribut : UNIQUE
Contrainte référentielle : REFERENCES <relation référencée>
[(<attribut référencé>)]
Valeur par défaut : DEFAULT <valeur>

Une contrainte de relation **concerne plusieurs attributs**

ex : Clé composée : PRIMARY KEY (nom_attribut [, nom_attribut]*)

Contrainte référentielle : FOREIGN KEY (nom_attribut [, nom_attribut]*)
REFERENCES nom de relation
[(nom_attribut [, nom_attribut]*)]

Exemple:
```
Create Table Livraison  (
        NumF INT NOT NULL references Fournisseur,
        NumP INT NOT NULL references Piece,
        DateLiv Date,
        Qté Integer ,
        Primary key (NumF, NumP, DateLiv)) ;
```

SQL - 2.4 : Requêtes de mise à jour - Insertion de tuples

- **INSERTION TUPLE à TUPLE**

INSERT INTO <nom_relation> [(nom_attribut [, nom_attribut]*)]
VALUES (valeur [, valeur]*) ;

Ex :

- **Les valeurs doivent être fournies dans l'ordre de déclaration des attributs de la liste ou, s'il n'y en n'a pas, du CREATE de la relation.**
- **Si la liste d'attributs est incomplète, les attributs non spécifiés sont insérés avec des valeurs NULL**

- **INSERTION D 'UN ENSEMBLE DE TUPLES 'CALCULES' A PARTIR DE TUPLES DE LA BASE DE DONNEES**

La clause VALUES est remplacée par un SELECT

EX :

SQL - 2.4 : Requêtes de mise à jour - Modification de tuples

UPDATE nom_relation

SET nom_attribut = <expression de valeur>

[, nom_attribut = <expression de valeur>]*

[WHERE <condition de recherche>];

La clause WHERE permet de sélectionner les tuples à mettre à jour

Ex :

<expression de valeur> peut être :

NULL, une constante ou une expression arithmétique contenant des attributs de la table à modifier

Ex :

SQL - 2.4 : Requêtes de mise à jour - Suppression de tuples

DELETE FROM <nom_relation> [WHERE <condition de recherche>];

Ex :

Pièce (NomP, Prix, Couleur, NomF)

Chapitre 3 : Algèbre Relationnelle - Opérateurs de base

C'est un langage de manipulation de données

- **Permet de définir comment interroger une Base de Données**
- **Facile à traduire en SQL (langage standard des Bases de Données)**
- **Objets manipulés : Les relations**
- **Six opérateurs de base**
  - **opérateurs ensemblistes**
  - **opérateurs spécifiques**

Algèbre Relationnelle : 3.1 L 'opérateur Union R1 ∪ R2

Définition : l'union est une opération portant sur deux relations R1 et R2 de même schéma consistant à construire une relation de même schéma R3 ayant pour tuples ceux appartenant à R1 ou à R2 ou aux deux.

Opérateur commutatif (R1 ∪ R2 = R2 ∪ R1)

EMPLOYE1	(NoEmp,	Nom,	Année,	NoDep)
	1045	Dupond	1978	03
	2067	Dupont	1965	06
	0456	Martin	1981	03

EMPLOYE2	(NoEmp,	Nom,	Année,	NoDep)
	1045	Dupond	1978	03
	0278	Martin	1987	05
	0789	Blanc	1981	06

EMPLOYE = EMPLOYE1 ∪ EMPLOYE2

EMPLOYE

Algèbre Relationnelle : 3.2 L 'opérateur Différence R1 - R2

Définition : la différence est une opération portant sur deux relations R1 et R2 de même schéma consistant à construire une relation de même schéma R3 ayant pour tuples ceux appartenant à R1 et n'appartenant pas à R2.

Opérateur non commutatif (R1 - R2 ≠ R2 - R1)

EMPLOYE1	**(NoEmp,**	**Nom,**	**Année,**	**NoDep)**
	1045	**Dupond**	**1978**	**03**
	2067	**Dupont**	**1965**	**06**
	0456	**Martin**	**1981**	**03**

EMPLOYE2	**(NoEmp,**	**Nom,**	**Année,**	**NoDep)**
	1045	**Dupond**	**1978**	**03**
	0278	**Martin**	**1987**	**05**
	0789	**Blanc**	**1981**	**06**

EMPLOYE3 = EMPLOYE1 - EMPLOYE2

EMPLOYE3

Algèbre Relationnelle : 3.3 L 'opérateur Intersection R1 ∩ R2

Définition : l'intersection est une opération portant sur deux relations R1 et R2 de même schéma, consistant à construire une relation de même schéma R3 ayant pour tuples ceux appartenant à la fois à R1 et à R2.

Opérateur commutatif (R1 ∩ R2 = R2 ∩ R1)

EMPLOYE1	**(NoEmp,**	**Nom,**	**Année,**	**NoDep)**
	1045	**Dupond**	**1978**	**03**
	2067	**Dupont**	**1965**	**06**
	0456	**Martin**	**1981**	**03**

EMPLOYE2	**(NoEmp,**	**Nom,**	**Année,**	**NoDep)**
	1045	**Dupond**	**1978**	**03**
	0278	**Martin**	**1987**	**05**
	0789	**Blanc**	**1981**	**06**

EMPLOYE4 = EMPLOYE1 ∩ EMPLOYE2

EMPLOYE4

Algèbre Relationnelle : 3.4 L'opérateur Produit cartésien R1 × R2

Définition : le produit cartésien est une opération portant sur deux relations R1 et R2 consistant à construire une relation R3 ayant pour schéma la concaténation de celui de R1 avec celui de R2 et pour tuples toutes les combinaisons des tuples de R1 et R2.

Opérateur commutatif (R1 × R2 = R2 × R1)

Employé3	(NoEmp	Nom	Année	NoDep)
	2067	Dupont	1965	06
	0456	Martin	1981	03

Département	(NumD	Intitulé	Taille)
	03	Comptabilité	6
	06	Informatique	10

Res = Employé3 × Département

Algèbre Relationnelle : 3.5 L 'opérateur Projection $\Pi_{A1,\ldots,Ak}(R1)$

Définition : la projection est une opération portant sur une relation R1 et sur un ensemble d'attributs A1, A2, …, Ak de R1 qui produit une relation R2 réduite aux attributs mentionnés en opérande.

$$R2 = \Pi_{A1,\ldots,Ak}(R1)$$

Le schéma de R2 est inclus dans le schéma de R1. Les tuples en double sont supprimés.

EMPLOYE

NoEmp	Nom	Année	NoDep
1045	Dupond	1978	03
2067	Dupont	1965	06
0456	Martin	1981	03
0278	Martin	1987	05
0789	Blanc	1981	06

AN-EMBAUCH = $\Pi_{Nom, Année}$(EMPLOYE)

DEP-EMP = Π_{NoDep}(EMPLOYE)

Algèbre Relationnelle : 3.6 L 'opérateur Restriction $\sigma_{prédicat}(R1)$

Définition : la restriction est une opération portant sur une relation R1 produisant une relation R2 de même schéma ayant pour tuples ceux de R1 vérifiant la condition (prédicat) précisée en opérande.

- **Le schéma de R2 est identique au schéma de R1**

- **Les conditions de base sont de la forme :**
  <attribut> <opérateur> <valeur>
  avec opérateur $\in \{ =, \neq, <, \leq, >, \geq \}$

Il est possible de composer plusieurs conditions de base à l'aide des opérateurs booléens de disjonction ($\vee$, OU), conjonction ($\wedge$, ET), négation ($\neg$, NON).

Algèbre Relationnelle : 3.6 L 'opérateur Restriction $\sigma_{prédicat}(R1)$ (exemple)

Liste des employés du département 03.

DEP03 = $\sigma_{NoDep = 03}$ (EMPLOYE)

NoEmp	Nom	Année	NoDep
1045	Dupond	1978	03
0456	Martin	1981	03

Il est possible de composer plusieurs conditions de base à l'aide des opérateurs booléens de disjonction ($\vee$, OU), conjonction ($\wedge$, ET), négation ($\neg$, NON).

Liste des employés du département 03 embauchés avant l'année 1980

ANCIEN = $\sigma_{Année<1980 \wedge NoDep = 03}$ (EMPLOYE)

NoEmp	Nom	Année	NoDep
1045	Dupond	1978	03

Chapitre 4 : SQL - requêtes mono-relations

SELECT * ou < liste d 'attributs à afficher>

FROM < Nom des relations utilisées>

[WHERE < condition >] ;

Employé (NoEmpl, NomE, Année, TélE, Nodept)
Fournisseur (NomF, VilleF, AdresseF)
Pièce (NomP, Prix, Couleur)

Ex1: Liste des noms et villes des fournisseurs

- **Elimination des tuples en double dans le résultat : mot-clé DISTINCT**

Ex2: Liste de toutes les villes des fournisseurs

SQL - 4.1 : structure de base des requêtes

SELECT correspond à l 'opérateur de projection sur la liste d 'attributs demandée; il peut aussi être suivi de fonctions d'attributs (vu dans la suite).

FROM indique les relations concernées par la requête;

WHERE précise une condition et correspond à l 'opérateur de sélection.

Exemple : Liste des noms des fournisseurs habitant Paris

```
SELECT
FROM
WHERE
```

Le symbole * permet de sélectionner tous les attributs de la relation

SQL - 4.2 : expression de sélection du WHERE

La condition dans WHERE peut être :

- **une expression logique avec des opérateurs NOT, AND, OR.**

- **une comparaison avec des opérateurs : =, <>, >,<, >=, <=**

- **un test d'intervalle BETWEEN permettant de vérifier si la valeur d'un attribut est comprise entre deux constantes**

- **une comparaison de texte LIKE permettant de vérifier si un attribut de type chaîne de caractères contient une ou plusieurs sous-chaînes; _ remplace un caractère, % remplace une chaîne de caractères,**

SQL - 4.2 : expression de sélection du WHERE

- **un test de valeur nulle NULL signifiant que la valeur est inconnue,**

- **un test d'appartenance IN qui permet de vérifier si la valeur d 'un attribut appartient à une liste de constantes,**

Chapitre 5 : Expression de requêtes multi-relations

5.1 : L 'opérateur de Jointure R1 $\triangleright\triangleleft_{\text{prédicat}}$ R2 de l 'algèbre relationnelle

Définition : la jointure est une opération portant sur deux relations R1 et R2 consistant à construire une relation R3 ayant pour schéma la concaténation de celui de R1 et de R2 et pour tuples les tuples du produit cartésien de R1 par R2 vérifiant le prédicat de jointure.

- **Schéma de R3 : concaténation des schémas de R1 et R2**
  (aucune contrainte sur les schémas R1 et R2)

- **Opérateur commutatif : R1 $\triangleright\triangleleft_{\text{prédicat}}$ R2 = R2 $\triangleright\triangleleft_{\text{prédicat}}$ R1**

- **Prédicat : <attribut1> <opérateur> <attribut2>**
  où attribut1 $\in$ R1 et attribut2 $\in$ R2
  avec opérateur $\in \{ =, \neq, <, \leq, >, \geq \}$

Il est possible de composer plusieurs conditions de base à l'aide des opérateurs booléens de disjonction ($\vee$, OU), conjonction ($\wedge$, ET), négation ($\neg$, NON).

5.1 : Equi-JOINTURE

- **Equi-Jointure : critère de jointure sur attributs de types comparables avec le comparateur égalité**

Employe

N° empl	Nom_e	Ville_e	Age_e	N° chef_e
141	Dupond	Paris	40	500
36	Durand	Tours	40	500
251	Parent	Agen	25	600

Chef

N° chef	Nom_c	Age_c
500	Albert	50
523	Durieux	35
600	Jacquet	40

Employe ⋈ Chef
(N° chef_e = N° chef)

N° empl	Nom_e	Ville_e	Age_e	N° chef_e	N° chef	Nom_c	Age_c

5.1 : θ-JOINTURE

- **θ-JOINTURE : critère de jointure sur attributs de types comparables avec un comparateur différent de l'égalité (≠, >, <, ≥, ≤, …)**

Employé

N° empl	Nom_e	Ville_e	Age_e	N° chef_e
141	Dupond	Paris	40	500
36	Durand	Tours	40	500
251	Parent	Agen	25	600
27	Barbier	Paris	53	500
125	Lefevre	Paris	30	523
208	legrand	Evry	39	523

Chef

N° chef	Nom_c	Age_c
500	Albert	50
523	Durieux	35
600	Jacquet	40

Employe ⋈ Chef
(N° chef_e = N° chef ∧ Age_e > Age_c)

N° empl	Nom_e	Ville_e	Age_e	N° chef_e	N° chef	Nom_c	Age_c

5.1 : Jointure naturelle (SQL92)

Prédicat : égalité des attributs communs (même nom) aux deux relations

Schéma de R3 : on ne garde qu'une seule fois les attributs communs

EMPLOYE

NoEmp	Nom	Année	NoDep
1045	Dupond	1978	03
2067	Dupont	1965	06
0456	Martin	1981	03
0278	Martin	1987	05

DEPARTEMENT

NoDep	Intitulé	Taille	NoResp
03	Compta	6	0456
06	Info	10	1249
05	Achats	3	0278

INFO-EMP = EMPLOYE ⋈ DEPARTEMENT

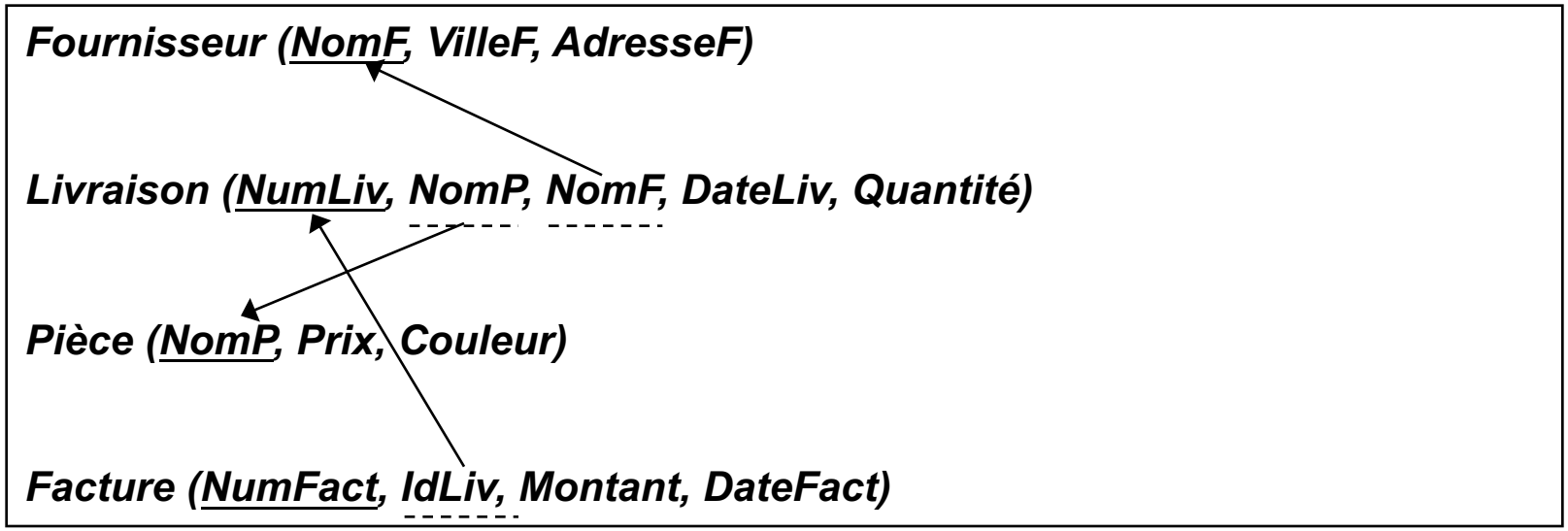
JN

NoEmp	Nom	Année	NoDep	Intitulé	Taille	NoResp
1045	Dupond	1978	03	Compta	6	0456
2067	Dupont	1965	06	Info	10	1249
0456	Martin	1981	03	Compta	6	0456
0278	Martin	1987	05	Achats	3	0278

Remarque : La jointure n'est pas un opérateur de base de l 'algèbre relationnelle

$$R1 \bowtie_{prédicat} R2 = \sigma_{prédicat} (R1 \times R2)$$

5.2 : SQL - expression des jointures



SQL-89 : on sélectionne des tuples du produit cartésien des relations citées dans la clause FROM

- **le WHERE permet d'exprimer les prédicats de jointure.**

Noms et ville des fournisseurs qui ont des livraisons, avec le nom des pièces livrées

```
SELECT Fournisseur.NomF, VilleF, NomP
FROM Fournisseur, Livraison
WHERE Fournisseur.NomF = Livraison. NomF ;
```

5.2 : SQL - expression des jointures

- **Avec une restriction dans la clause WHERE :**

  Noms des fournisseurs de Melun qui ont des livraisons et date de ces livraisons

- **Elimination des doublons :**

- **Utilisation de variables :**

```
SELECT DISTINCT DateLiv, F.NomF
FROM Fournisseur F, Livraison L
WHERE F. NomF = L.NomF;
```

5.2 : SQL - expression des jointures

SQL-92 : spécification de la jointure dans la clause FROM

• Jointure naturelle

Noms et numéros de livraison des fournisseurs de Paris qui ont livré

```
SELECT DISTINCT NomF, NumLiv
FROM Fournisseur NATURAL JOIN Livraison
WHERE VilleF = ‘ Paris ’;
```

Ville des fournisseurs qui ont livré des pièces rouges

```
SELECT DISTINCT VilleF
FROM Fournisseur NATURAL JOIN Livraison  NATURAL JOIN Pièce
WHERE Couleur = ‘ rouge ’;
```

• Equi-jointure en spécifiant les attributs de jointure (qui ne sont mis qu’une seule fois dans le résultat)

```
SELECT DISTINCT NomF, NumLiv
FROM Fournisseur JOIN Livraison USING (NomF)
WHERE VilleF = ‘ Paris ’;
```

• Equi-jointure en spécifiant le critère de jointure

Date et numéro des factures des livraisons du fournisseur 'Carrefour'

```
SELECT NumFact, DateFact,
FROM Facture F JOIN Livraison L ON F.IdLiv = L.NumLiv
WHERE NomF = 'Carrefour' ;
```

5.2 : SQL - expression des jointures externes

SQL-92 : on garde dans le résultat les tuples qui ne joignent pas (d'une relation, de l 'autre, ou des deux)

livre

Titre	N°auteur
La peste	56
L'assommoir	10

auteur

N°auteur	Nom
10	Zola
15	Tolkien

SELECT * FROM livre LEFT OUTER JOIN auteur ON livre.n° auteur = auteur.n° auteur ;

==>

Titre	Livre.N°auteur	Auteur.N°auteur	Nom
La peste	56	**Null**	**Null**
L'assommoir	10	10	Zola

SELECT * FROM livre RIGHT OUTER JOIN auteur ON livre.n° auteur = auteur.n° auteur ;

==>

Titre	Livre.N°auteur	Auteur.N°auteur	Nom
Null	**Null**	15	Tolkien
L'assommoir	10	10	Zola

SELECT * FROM livre FULL OUTER JOIN auteur ON livre.n° auteur = auteur.n° auteur ;

==>

Titre	Livre.N°auteur	Auteur.N°auteur	Nom
Null	**Null**	15	Tolkien
L'assommoir	10	10	Zola
La peste	56	**Null**	**Null**

5.2 : SQL - Produit cartésien

Produit cartésien A × B : sélectionne tous les couples possibles de A et B

Fournisseur

NomF	VilleF	AdresseF
Carrefour	Melun	Rue Dupond
Auchan	Paris	La Bastille

Livraison

NumLiv	NomP	NomF	DateLiv	Quantité
10	Lait	Carrefour	10/10/18	10
5	Pain	Auchan	25/03/19	29
27	Pizza	Carrefour	10/10/18	24

SELECT * FROM Fournisseur, Livraison;

NomF	VilleF	AdresseF	NumLiv	NomP	NomF	DateLiv	Quantité
Carrefour	Melun	Rue Dupond	10	Lait	Carrefour	10/10/18	10
Carrefour	Melun	Rue Dupond	5	Pain	Auchan	25/03/19	29
Carrefour	Melun	Rue Dupond	27	Pizza	Carrefour	10/10/18	24
Auchan	Paris	La Bastille	10	Lait	Carrefour	10/10/18	10
Auchan	Paris	La Bastille	5	Pain	Auchan	25/03/19	29
Auchan	Paris	La Bastille	27	Pizza	Carrefour	10/10/18	24

5.3 : SQL - fonctions ensemblistes dans la clause WHERE

Le résultat d 'une requête SELECT est un *ensemble* de tuples. On peut utiliser ce résultat, avec des fonctions ensemblistes, dans un prédicat de la clause WHERE

- **nom_attribut [NOT] IN (< sous-requête >)**

évalué à VRAI si nom_attribut appartient à l 'ensemble des résultats de <sous-requête>

Ex: Lister les noms des pièces qui n'ont pas été livrées

```
SELECT NomP
FROM  Pièce P
WHERE NomP NOT IN (SELECT NomP FROM Livraison);
```

Ex : Nom et ville des fournisseurs qui ont livré la pièce ' XXA2 '

```
SELECT NomF, VilleF
FROM Fournisseur
WHERE NomF IN ( SELECT NomF
                FROM Livraison
                WHERE NomP = ' XXA2 ' );
```

équivaut à

```
SELECT DISTINCT F. NomF, VilleF
FROM Fournisseur F, Livraison L
WHERE ( F. NomF = L. NomF )
        AND ( L. NomP = ' XXA2 ' );
```

5.3 : SQL - fonctions ensemblistes dans la clause WHERE

Autre exemple : Lister les couleurs des pièces livrées par les fournisseurs habitant Paris

```
SELECT DISTINCT Couleur
FROM Pièce
WHERE NomP IN (SELECT NomP
                    FROM Livraison
                    WHERE NomF IN (SELECT NomF
                                        FROM Fournisseur
                                        WHERE VilleF = ‘ Paris ’));
```

équivaut à

```
SELECT DISTINCT Couleur                                          (SQL89)
FROM Pièce P, Livraison L, Fournisseur F
WHERE  P. NomP = L. NomP  AND  L. NomF =F. NomF
           AND VilleF = ‘ Paris ’;
```

```
SELECT DISTINCT Couleur                                          (SQL92)
FROM Fournisseur NATURAL JOIN Livraison  NATURAL JOIN Pièce
          WHERE VilleF = ‘ Paris ’;
```

5.3 : SQL - fonctions quantifiées dans le WHERE

- **nom_attribut = | <= | < | > | >= … ANY (< sous-requête >)**
  évalué à VRAI s 'il existe au moins un élément de <sous-requête>
  qui vérifie la comparaison

  Ex : Donner les noms et villes des fournisseurs ayant livré la pièce AAX2

```
SELECT  NomF, VilleF
 FROM Fournisseur
 WHERE NomF = ANY (SELECT NomF
                   FROM Livraison
                   WHERE NomP ='AAX2');
```

- **nom_attribut = | <= | < | > | >= … ALL (< sous-requête >)**
  évalué à VRAI si pour tout élément de < sous- requête >,
  l 'opérateur de comparaison retourne VRAI

  Ex : Lister la plus grande quantité livrée

```
SELECT *
FROM Livraison
WHERE Quantité >= ALL (SELECT Quantité
                       FROM Livraison);
```

5.3 : SQL - fonctions quantifiées dans le WHERE

- **[NOT] EXISTS (< sous-requête >)**

évalué à VRAI si <sous-requête> retourne au moins un élément

Ex: Lister les pièces qui n'ont pas été livrées

```
SELECT NomP
FROM  Pièce P
WHERE NOT EXISTS (SELECT *
                  FROM Livraison
                  WHERE Livraison.NomP = P.NomP);
```

5.4 : SQL - expression des opérateurs ensemblistes

- **< requête > UNION < requête >**
- **< requête > INTERSECT < requête >**
- **< requête > EXCEPT < requête >**

les 2 requêtes doivent avoir le même nombre d'attributs avec des types comparables dans la clause SELECT

Ex: Nom des fournisseurs qui livrent la pièce SSZ4 mais pas la pièce A2

```
(SELECT NomF                    équivaut à    SELECT NomF
FROM Livraison                                FROM Livraison
WHERE  NomP = 'SSZ4')                         WHERE NomP = 'SSZ4'
 EXCEPT                                         AND NomF NOT IN (SELECT NomF
(SELECT NomF                                                     FROM Livraison
FROM Livraison                                                   WHERE NomP = 'A2');
WHERE NomP = 'A2');
```

5.5 : SQL - fonctions de calcul

Une fonction de calcul est une fonction qui s'applique sur un ensemble de tuples et qui renvoie une valeur unique

• Calcul sur les valeurs prises par un attribut

AVG (DISTINCT | ALL nom_attribut) = moyenne des valeurs prises par *nom_attribut*
SUM (DISTINCT | ALL nom_attribut) = somme des valeurs prises par *nom_attribut*

***nom_attribut* doit être de type entier ou réel**

Ex : La quantité totale livrée des ' Aspirateur '

Fournisseur (NomF, VilleF, AdresseF)

Livraison (NumLiv, NomP, NomF, DateLiv, Quantité)

Pièce (NomP, Prix, Couleur)

5.5 : SQL - fonctions de calcul

MAX (nom_attribut) = maximum des valeurs prises par *nom_attribut*
MIN (nom_attribut) = minimum des valeurs prises par *nom_attribut*
***nom_attribut* doit être de type entier, réel, caractère ou date**

Ex : La livraison maximale de ‘ Aspirateur ’

- **Comptage du nombre de tuples**

COUNT(* | [ALL | DISTINCT nom_attribut]) =
nombre de valeurs de l’ensemble résultat
- dans le cas de * : y compris les valeurs nulles
- avec DISTINCT : sans les doublons
- avec ALL : avec les doublons

Ex : Nombre de livraisons de la pièce de nom ‘ xxa1 ’

Ex : Nombre de couleurs différentes dans les pièces

5.5 : SQL - fonctions de calcul

- **Expression arithmétique**

  Exemple :

```
SELECT NumLiv, NomP, Quantité * Prix
FROM Livraison Natural Join Pièce
```

Les fonctions de calcul peuvent être utilisées dans la clause WHERE, elles portent alors sur un SELECT

EX : Donner les noms de pièce ayant fait l 'objet d 'au moins 10 livraisons

```
SELECT P.NomP
FROM Piece P
WHERE (SELECT Count (*)
        FROM Livraison L
        WHERE P.NomP = L.NomP) >= 10
```

5.6 : SQL - tri du résultat

La clause ORDER BY permet de trier le résultat de la requête, en fournissant la liste des attributs (ou de n° s de colonnes) sur lesquels effectuer le tri et en spécifiant le sens du tri (ascendant ou descendant).

EMPLOYE (NumEmp, NomEmp, Salaire, Fonction, NumDepartement)

Ex 1 : Liste des employés triée par ordre alphabétique sur le nom

```
SELECT *
FROM Employe
ORDER BY NomEmp ASC
```

Ex 2 : Liste des employés triée sur le salaire par ordre croissant et sur le nom par ordre décroissant

```
SELECT *
FROM Employe
ORDER BY Salaire ASC, NomEmp DESC
```

SQL - Résumé

Forme simple d'une requête SQL

4 **SELECT** attributs résultats [avec fonctions]

1 **FROM** relations utilisées

2 [**WHERE** condition de sélection]

3 [**ORDER BY** critère de tri du résultat]

Chapitre 6 : SQL - Les Vues (1)

- **Objectif**

- **Définition d'une Vue (View)**

  Relation virtuelle dont le schéma et le contenu sont dérivés de la base de données réelle par un ensemble de questions.

  Une vue est donc une relation déduite d'une base de données, par composition des relations de la base.

- **Moyen**

- **Problèmes**

SQL - Création et destruction d'une vue

- **Création**

  CREATE VIEW < nom_vue > [(Liste d'Attributs)]
  AS <Requête SQL SELECT >
  [WITH CHECK OPTION]

  – **<nom_vue>**

  – **[(Liste d'Attributs)] : (clause optionnelle)**

  – **La clause WITH CHECK OPTION**

- **Destruction**
  – **DROP VIEW <nom de vue>**

SQL - Exemples de création de vues

CLIENT (NumCli, RaisonSociale, VilleClient, Adresse Livraison, CA)

COMMANDE (NumCom, Numcli, NumPro, DateCom, QtteCom)

PRODUIT (NumPro, Libellé, Prix)

Create view BonClient (NumCli, NomCli, VilCli, CA)

As

Create view CommandeDuJour (NumCli, NumPro, QtteCom)

As

Create view CommandeDeClient (NumCom, NumCli, RaisonSociale, VilleClient, Adresse Livraison)

As

Interrogation de vues

- **Transparence pour l'utilisateur**
- **Objectif : Simplifier les requêtes utilisateurs**
- **Exemples :**
  - **Noms des bons clients de Paris**
  - **Numéros et noms des bons clients qui ont commandé aujourd'hui le produit 10**

- **Modification de question**

  Mécanisme consistant à modifier une question au niveau du source en remplaçant certaines relations du FROM par leurs sources et en enrichissant les conditions de la clause WHERE pour obtenir le résultat de la question initiale.

- **Peut être effectuée au niveau du source ou par concaténation d'arbres algébriques (cf cours 7)**

- **Concrètement**

Mise à jour des Vues

- **Vue qu'on peut mettre à jour**
  - **Vue comportant suffisamment d'information pour permettre un report des mises à jour dans la base sans ambiguïté.**

  Ex :

  est traduite par :

- **Les mises à jour ne sont possibles que sous certaines conditions**

Clause ‘With Check Option’

Si la clause With Check Option est présente alors les opérations Insert et Update seront testées de manière à garantir que le nouveau tuple satisfait les conditions définissant la vue. Dans le cas contraire, l’opération est rejetée.

Utilisation de la clause Insert

Utilisation de la clause Update

Intérêt des vues

- **Confidentialité**

  moyen de restreindre les données de la base à l 'utilisateur
  par exemple, un utilisateur n'ayant un accès qu'à une vue de la relation Employé dans laquelle le salaire est omis

- **Simplification et adéquation**

  dans une base multi-utilisateurs, chacun est concernée par un sous ensemble de données.
  La définition de données pour des types d 'utilisateurs permet de leur fournir ce dont ils ont besoin uniquement avec des noms d 'attributs qui leur conviennent

- **Lisibilité**

  une vue produite à partir de plusieurs tables est plus facilement compréhensible

Chapitre 7 : Les Langages Algébriques

L'application d'un opérateur de l'algèbre relationnelle sur une ou deux relations produit en résultat une nouvelle relation qui peut alors être, à son tour, opérande d'un opérateur de l'algèbre relationnelle.

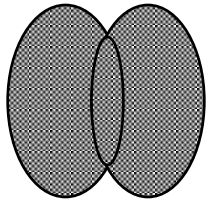
Langage algébrique :

Opérateurs de l'algèbre relationnelle :

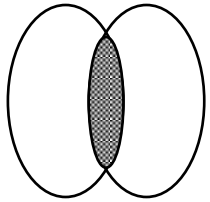
- **notations algébriques**
- **notations graphiques**
- **notations fonctionnelles**

Algèbre Relationnelle : Résumé

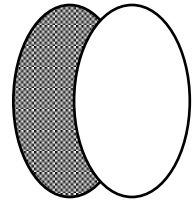
Résumé des principaux opérateurs



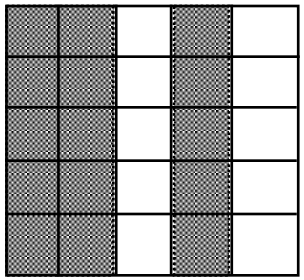
Union



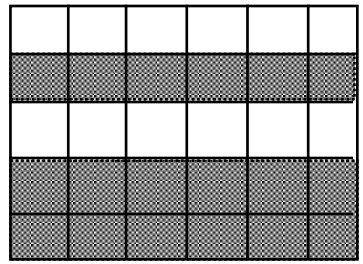
Intersection



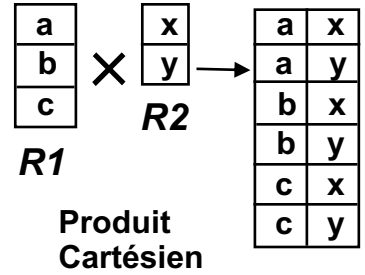
Différence



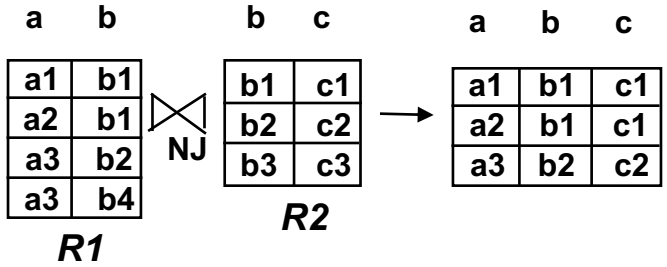
Projection



Restriction



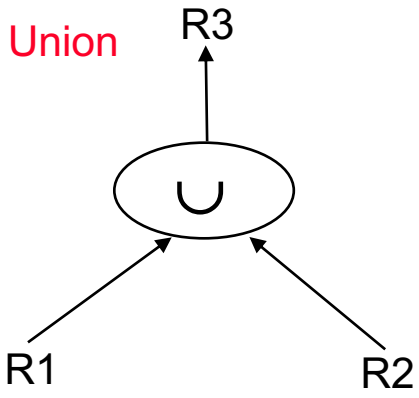
Produit Cartésien



Jointure naturelle

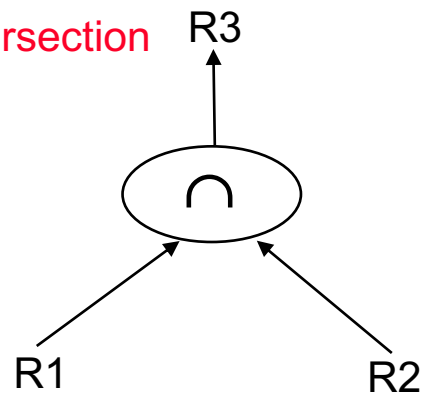
Algèbre Relationnelle : Différentes Notations (1)

Union



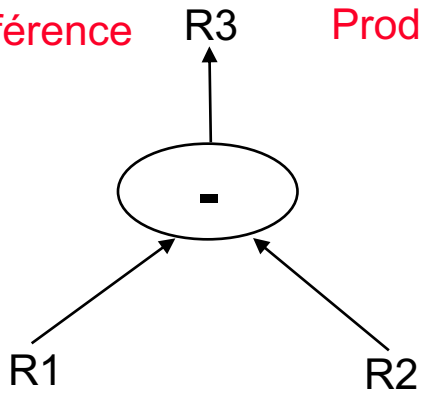
$R3 = R1 \cup R2$
R3 = Union(R1,R2)

Intersection



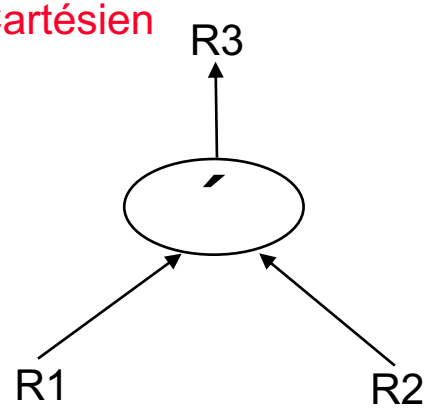
$R3 = R1 \cap R2$
R3 = Inter(R1,R2)

Différence



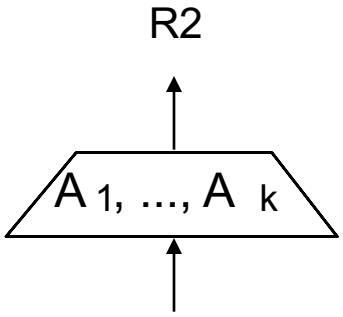
$R3 = R1 - R2$
R3 = Diff(R1,R2)

Produit Cartésien



$R3 = R1 \times R2$
R3 = Product(R1,R2)

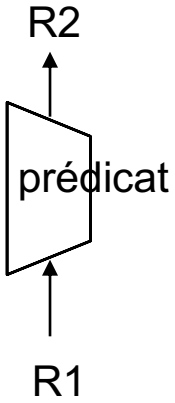
Projection



$R2 = \Pi_{A_1, ..., A_k}(R1)$

R2 = Project(R1/ $A_1, ..., A_k$)

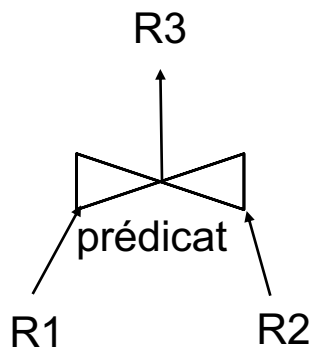
Restriction



R2 = σ(R1/prédicat)
R2 = Restrict(R1/prédicat)

Algèbre Relationnelle : Différentes Notations

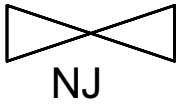
Jointure



R3 = Join (R1,R2/prédicat)

R3 = R1 $\rhd\lhd_{\textbf{prédicat}}$ R2

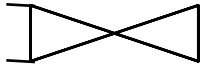
Jointure Naturelle



R3 = NatJoin (R1,R2)

Jointures Externes

Left Outer Join

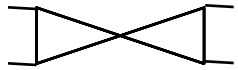


R3 = XJoin (*R1,R2/prédicat)

Right Outer Join

R3 = XJoin (R1,*R2/prédicat)

Full Outer Join



R3 = XJoin (*R1,*R2/prédicat)

Algèbre Relationnelle

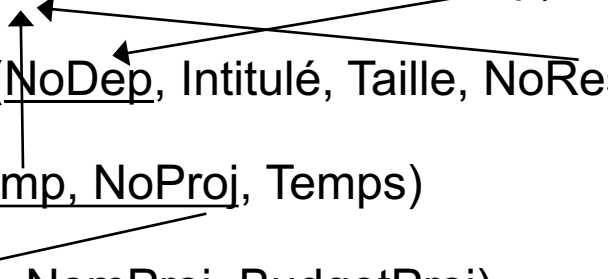
Soit la base de données constituée des relations suivantes :

Employé (NoEmp, Nom, Année, NoDep)

Département (NoDep, Intitulé, Taille, NoResp)

Travaille (NoEmp, NoProj, Temps)

Projet (NoProj, NomProj, BudgetProj)



- **Noms des employés du département 03**

$\Pi_{nom}(\sigma_{Nodep = 03}(\textbf{Employé}))$

Project(Restrict(Employé/NoDep=03)/Nom)

Algèbre Relationnelle

- **Nom du responsable du département Achats.**

Π_{nom} (Employé $\overset{\cdot}{\bowtie}_{NoResp = NoEmp}$ ($\sigma_{Intitulé=' Achats '}$(Département))

Project (Join (Employé, Restrict(Département/ Intitulé = 'Achats ') / NoResp = NoEmp) / Nom)

Algèbre Relationnelle

- **Numéros des employés qui ne sont pas des responsables de département**

Algèbre Relationnelle

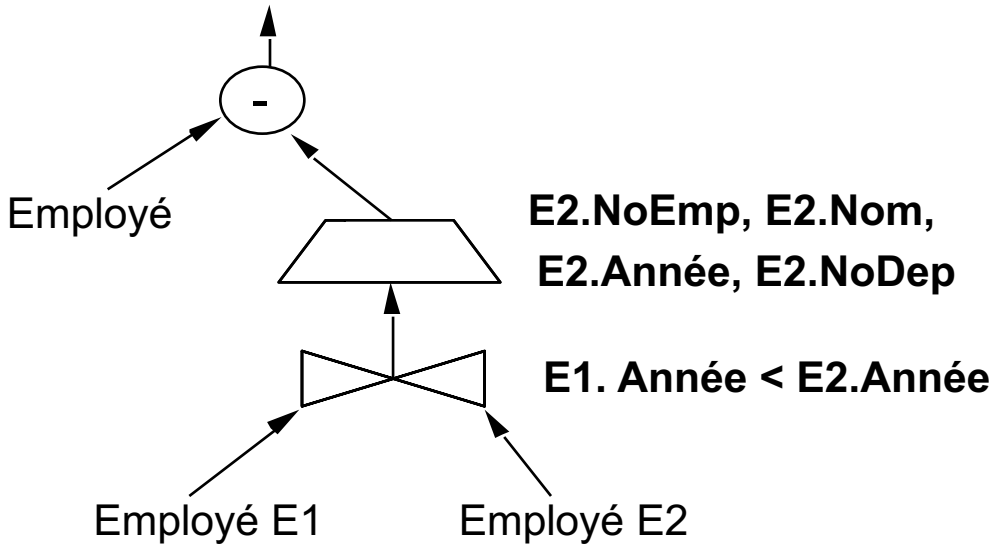
- **Numéros des Employés qui sont à la fois responsables d'un département et travaillent sur au moins un projet.**

Algèbre Relationnelle

- **Numéros des projets des employés du département Gestion**

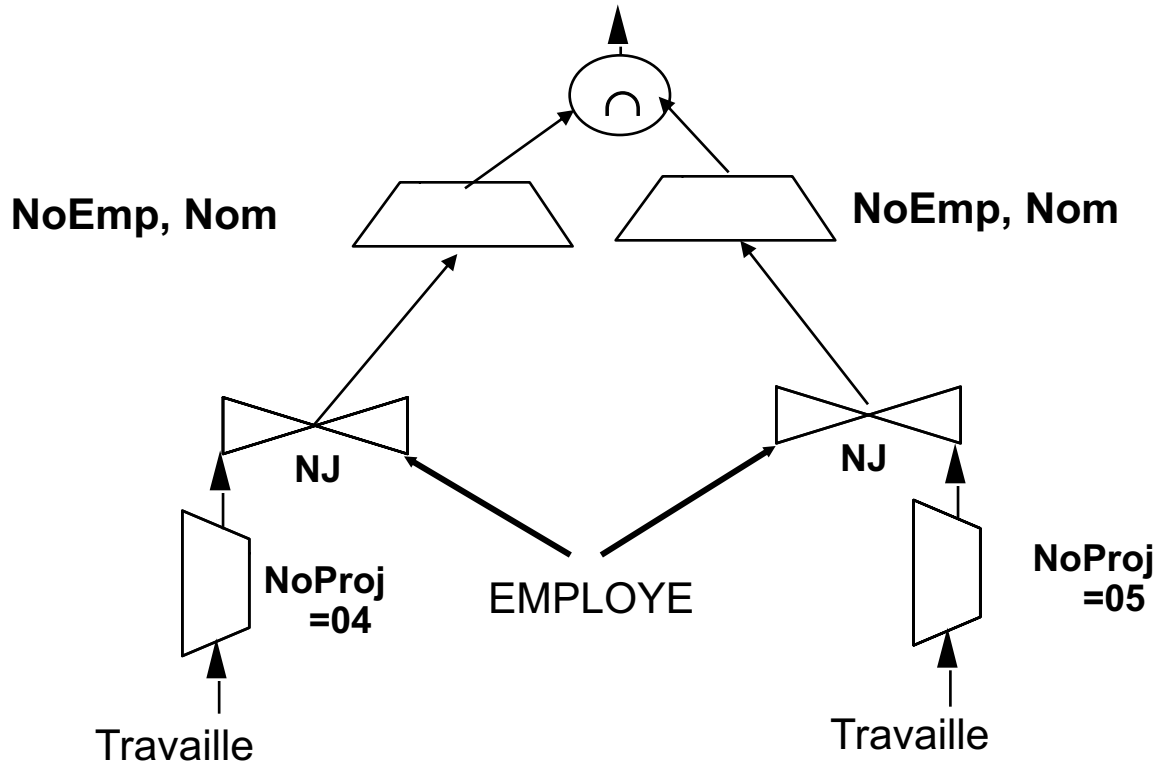
Algèbre Relationnelle

Informations concernant les plus anciens employés.



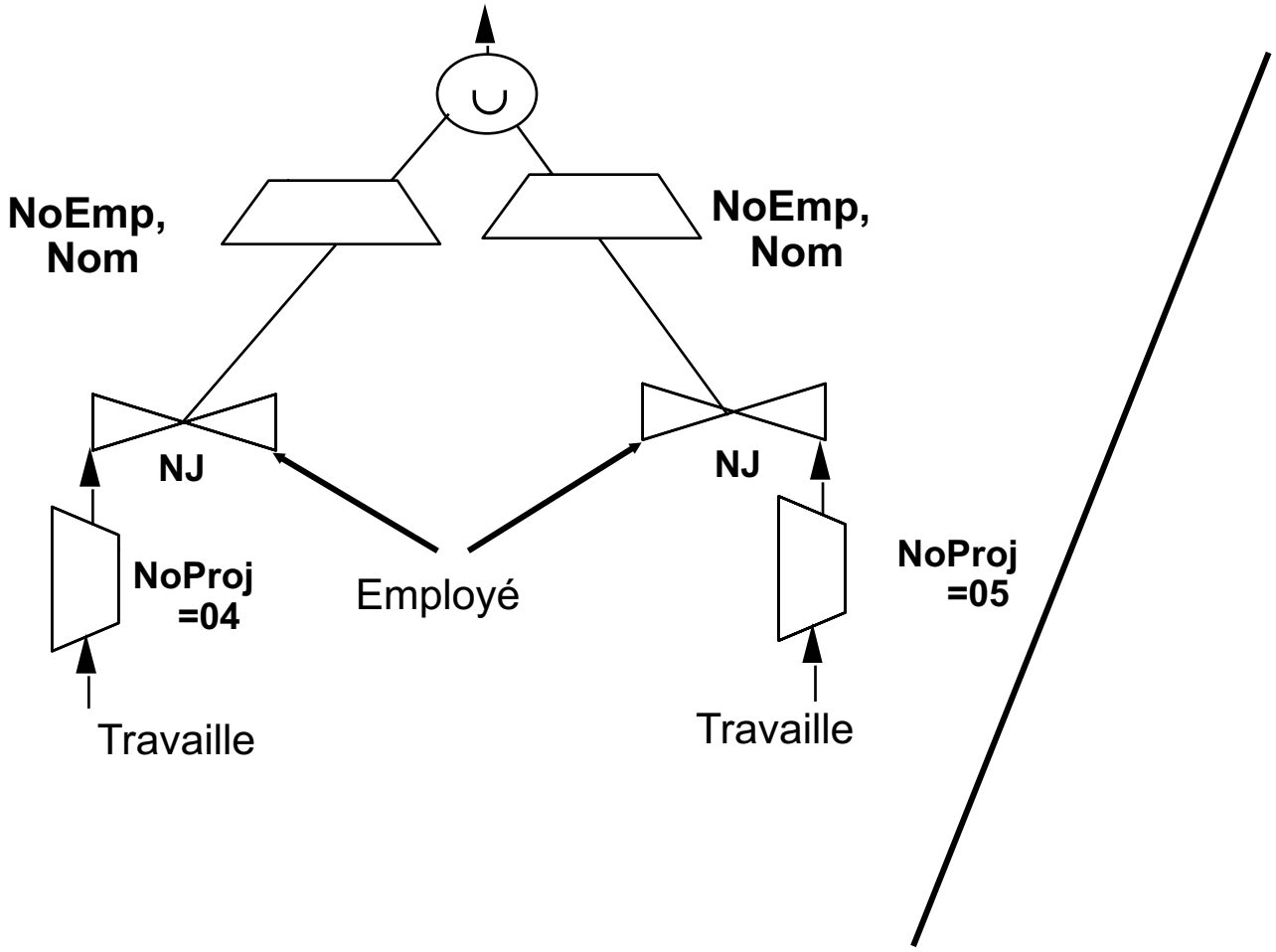
Algèbre Relationnelle

Les noms et numéros des employés qui travaillent sur le projet 04 et sur le projet 05.



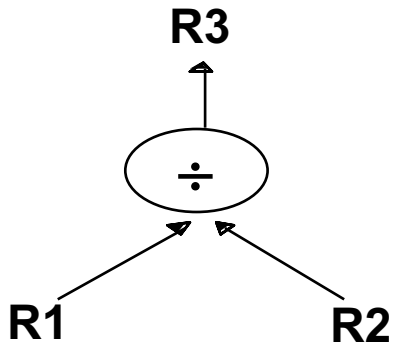
Algèbre Relationnelle

Les noms des employés qui travaillent sur le projet 04 ou sur le projet 05.



Algèbre Relationnelle : L 'opérateur Division R1 ÷ R2 ou Div(R1,R2)

Définition : la division est une opération portant sur deux relations R1 et R2 ayant des attributs en commun consistant à construire une relation R3 réduite aux attributs de R1 n'appartenant pas à R2 et ayant pour tuples ceux qui concaténés à tout tuple de R2 donnent un tuple de R1.



- schéma de R2 ⊂ schéma de R1
- schéma de R3 : schéma de R1 - schéma de R2
- Opérateur non commutatif

Algèbre Relationnelle : Division

Exemple : la liste des employés travaillant sur tous les projets

AFFECT = TRAVAILLE1 ÷ PROJET1

TRAVAILLE1

NoEmp	NoProj
1045	A15
1045	B18
1045	A20
0456	A20
0278	A15
0278	B18
0789	B18
0789	A20
0789	A15

÷

PROJET1

NoProj
A15
B18
A20

=

AFFECT

NoEmp
1045
0789

Algèbre Relationnelle: la division

Les numéros des employés qui travaillent sur tous les projets

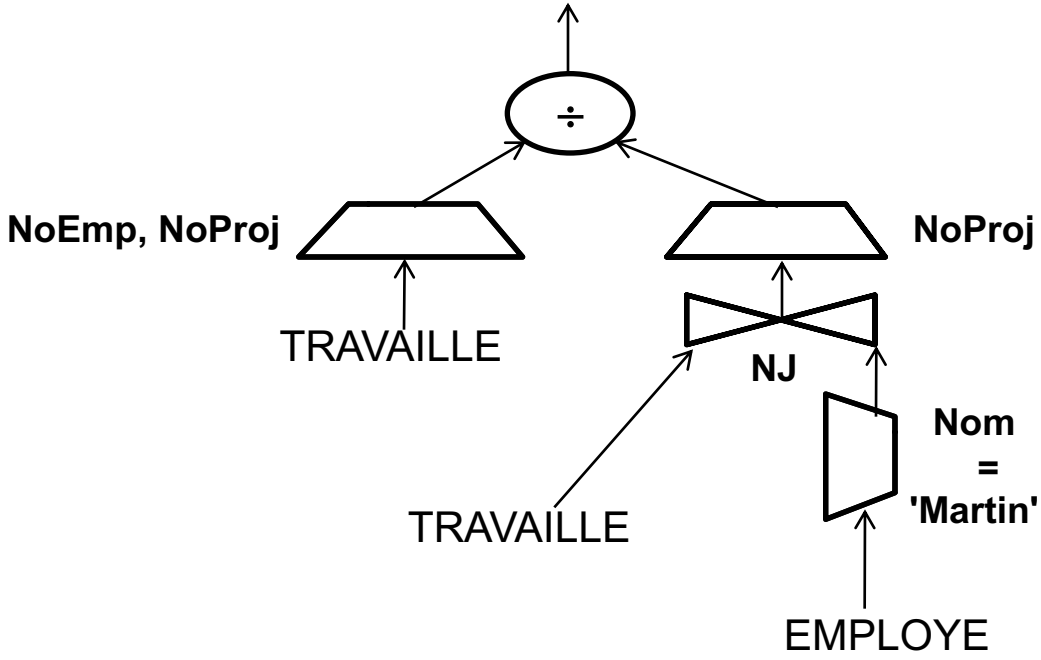
Div(Project(TRAVAILLE/ NoEmp, NoProj), Project(PROJET/ NoProj))

Algèbre Relationnelle : la division

Les numéros des employés qui travaillent sur les mêmes projets que l'employé 'Martin'

Z=Project(Join(TRAVAILLE,Restrict(EMPLOYE/Nom='Martin')/NoEmp=NoEmp)/NoProj)

Result=Div(Project(TRAVAILLE/ NoEmp, NoProj), Z)



Les *noms* des employés qui travaillent sur les mêmes projets que l'employé 'Martin' :

Algèbre Relationnelle : Division

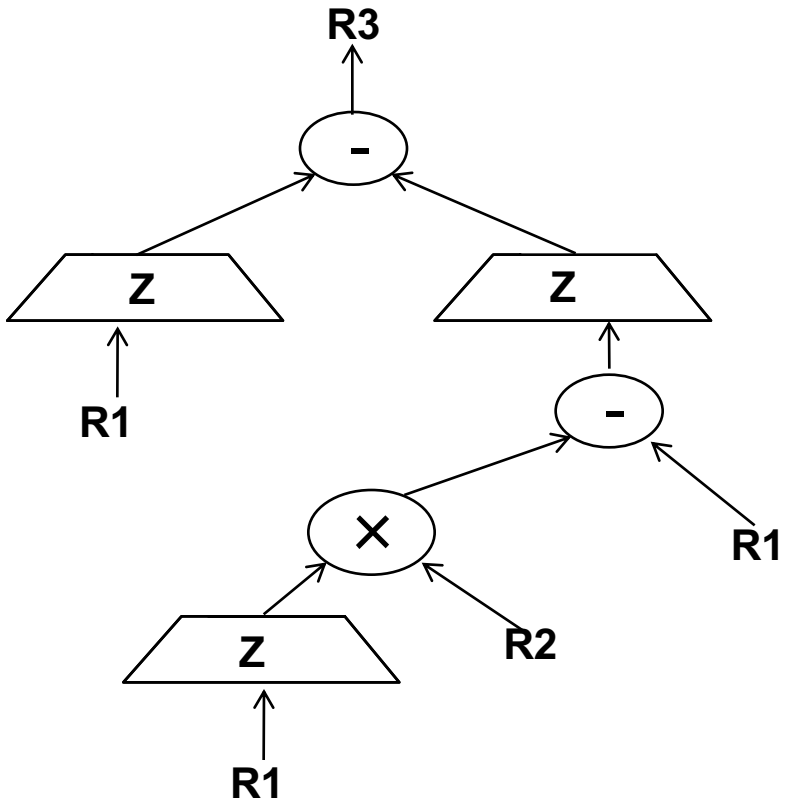
Remarque : la division n'est pas une opération de base et peut s'exprimer comme

R3(Z) = R1(X) ÷ R2(Y) = T(Z) - U(Z)

avec $Y \subseteq X$ **Z = X-Y**

T(Z) = $\Pi_Z(R1)$

U(Z) = $\Pi_Z((T \times R2) - R1)$



SQL - expression de la division

Trouver les noms des employés qui travaillent sur tous les projets :

Chapitre 8 : Les Langages Assertionnels – Le Calcul Relationnel de Tuples

Fondés sur la logique du premier ordre (calcul des prédicats)

Une base de données = un ensemble de propositions logiques représentant l'univers du discours (appelé aussi domaine d'interprétation)

Une requête = une formule du premier ordre à évaluer sur l'univers du discours (contenu de la base de données)

Une variable de cette formule = un tuple d'une relation

Calcul relationnel de tuples

Définition : langage formel d 'interrogation de données permettant d 'exprimer des questions à partir des formules bien formées dont les variables sont interprétées comme variant sur des tuples d 'une relation.

- **Les variables sont associées à des tuples**

Fournisseur (Numf, Nomf, Vilf)

X ∈ Fournisseur

est une variable de la forme

<Numf, Nomf, Vilf>

est une relation

La valeur de X peut être, par exemple, <7, 'Géant', 'Londres'>.

Calcul relationnel de tuples

- Les termes sont :
  - des constantes associées aux domaines :
  - des fonctions qui associent à un tuple une valeur d 'attribut X.A

  où X est une variable et A un attribut :

- Les prédicats sont :
  - des comparaisons avec des opérateurs =, >, <>, …`
  - des tests d 'appartenance à une relation

Calcul relationnel de tuples

La syntaxe d 'une requête est :

{ ((t1.A1, ... , t1.Ap), … , (tm.B1, ... , tm.Bq)) /

R1(t1) ∧ … ∧ Rm(tm)

∧ (Q t'1, …, Q t'n) (S1(t'1) ∧ … ∧ Sn(t'n)

∧ W(t1, …, tm, t'1, …, t'n)) }

- **Les ti (i variant de 1 à m) désignent les variables tuples qu'on veut retrouver. Les Aj sont des attributs de t1, les Bk sont des attributs de tm. ((t1.A1, ... , t1.Ap), … , (tm.B1, ... , tm.Bq)) représente le critère de projection.**
- **Les Ri sont les relations sur lesquelles les variables ti sont définies.**
- **W est une expression de prédicats (une restriction) qui sélectionne les tuples désirés.**
- **Q est un quantificateur (universel ∀ ou existentiel ∃)**
- **les t'j sont des variables tuples liées définies sur des relations Sj. Les Sj ne sont pas nécessairement distinctes entre elles ni distinctes des Ri.**

Calcul relationnel de tuples

Requêtes mono-relation:

1)liste des noms et des villes de tous les fournisseurs

2) liste des noms des fournisseurs habitant Londres

Expression de la jointure :

3) liste des pièces livrées avec les numéros de fournisseurs qui les livrent

Calcul relationnel de tuples

4) Liste des noms des fournisseurs qui ont livré la pièce n° 7

{ (F.Nomf) / Fournisseur(F) $\wedge$

$\exists$ L (Livraison(L) $\wedge$ ((L.Numf = F.Numf) $\wedge$ (L.Nump = 7)))}

explication :

- **le premier nom dans Fournisseur est « Géant »,**
- **son numéro correspondant est 3,**
- **Existe-t-il un L (tuple) dans Livraison tel que le Numf = 3 et le Nump = 7 ?**
- **Si oui, alors « Géant » fait partie du résultat.**

Calcul relationnel de tuples

5) Liste des noms des pièces livrées par le fournisseur n° 3

{ (P.Nomp) / Pièce(P) ∧

∃ L (Livraison(L) ∧ ((L.Nump = P.Nump) ∧ (L.Numf = 3)))}

6) Liste des noms des pièces livrées par le fournisseur de nom « Géant »

{ (P.Nomp) / Pièce(P) ∧ ∃ L ∃ F (Livraison(L) ∧ Fournisseur(F)

∧ ((L.Nump = P.Nump) ∧ (L.Numf = F.Numf) ∧ (F.Nomf = 'Géant ')))}

Calcul relationnel de tuples

7) Liste des fournisseurs qui livrent à la fois les pièces n° 2 et n° 7

{ (F.Numf) / Fournisseur(F) ∧

∃ L1 ∃ L2 (Livraison(L1) ∧ ((L1.Nump = 2) ∧ (L1.Numf = F.Numf)) ∧

Livraison(L2) ∧ ((L2.Nump = 7) ∧ (L2.Numf = F.Numf))))}

7 ') Liste des fournisseurs qui livrent les pièces n° 2 mais pas n° 7

… ∃ L1 (Livraison(L1) ∧ ((L1.Nump = 2) ∧ (L1.Numf = F.Numf)) ∧

¬∃ L2 (Livraison(L2) ∧ ((L2.Nump = 7) ∧ (L2.Numf = F.Numf)))

7 ' ') Liste des fournisseurs qui livrent les pièces n° 2 ou n° 7

plus simple - une seule variable

{ (F.Numf) / Fournisseur(F) ∧ ∃ L (Livraison(L) ∧ ((L.Nump = 2) ∨ (L.Nump = 7))

∧ (L.Numf = F.Numf)))}

Calcul relationnel de tuples

8) Liste des noms de fournisseurs qui ne livrent pas la pièce n° 7

$$\{ (F.Numf) / Fournisseur(F) \wedge$$
$$\forall L\ (Livraison(L) \Rightarrow ((L.Numf \neq F.Numf) \vee (L.Nump \neq 7))) \}$$

explication :

un tuple de Fournisseur étant fixé, pour tout tuple de Livraison :

- **ou bien le fournisseur ne fait pas de livraison,**
- **ou bien le n° de la pièce n 'est pas 7.**

Équivaut à :

$$\{ (F.Numf) / Fournisseur(F) \wedge$$
$$\neg (\exists L\ (Livraison(L) \wedge ((L.Numf = F.Numf) \wedge (L.Nump = 7)))) \}$$

Calcul relationnel de tuples

(lois de de Morgan)

$(A \Rightarrow B \Leftrightarrow \neg A \vee B)$ **$(\forall x\ P(x) \Leftrightarrow \neg\exists x\ \neg P(x))$**

$(\neg\ (A \vee B) = \neg A \wedge \neg B)$ **$(\neg\neg A = A)$**

$\{\ (\ F.Numf)\ /\ Fournisseur(F) \wedge \forall\ L\ (Livraison(L) \Rightarrow (\ (\ L.Numf \neq F.Numf) \vee (\ L.Nump \neq 7)\ )\ )\}$

$\{(\ F.Numf)\ /\ Fournisseur(F) \wedge \neg\ (\ \exists\ L\ (Livraison(L) \wedge ((\ L.Numf = F.Numf) \wedge (\ L.Nump = 7\ )\ )\ )\ )\}$

Calcul relationnel de tuples

Expression de la division :

9) Liste des noms des fournisseurs qui ont livré toutes les pièces

{ (F.Nomf) / Fournisseur(F) ∧ ∀ P ∃ L (Pièce(P) ∧ Livraison(L)

∧ ((P.Nump = L.Nump) ∧ (L.Numf = F.Numf)))}

Fournisseur

Numf	Nomf
10	A
11	B

Livraison

Numf	Nump	Qté
10	1	3
10	4	7
11	1	6

Pièce

Nump	Nomp
1	X
4	Y

Traduction en SQL

Calcul relationnel de tuples

10) liste des numéros de fournisseurs qui livrent au moins toutes les pièces livrées par le fournisseur n° 2

$$\{ (F.Numf) / Fournisseur(F) \wedge \forall P \ \exists L1 \ \exists L2$$

$$(Pièce(P) \wedge Livraison(L1) \wedge Livraison(L2) \wedge$$

$$(((L1.Numf = 2) \wedge (L1.Nump = P.Nump)) \Rightarrow$$

$$((L2.Numf = F.Numf) \wedge (L2.Nump = P.Nump)))) \ \}$$