ICPJ, 21928, CNAM, Paris
BASES DE DONNÉES
Relationnelles

M. Scholl et D. Vodislav

(scholl|vodislav)@cnam.fr

2003/2004

INTRODUCTION

Objectif

OBJECTIF:

Comprendre et Maitriser la technologie relationnelle

BIBLIOGRAPHIE

Ouvrages en français

1. P. Rigaux, *Cours bases de données*, cedric/cnam.fr/vertigo voir à support de cours.
2. Date C.J, *Introduction aux Bases de Données*, Vuibert, 970 Pages, Janvier 2001

Ouvrages en anglais

1. R. Ramakrishnan et J. Gehrke, *DATABASE MANAGEMENT SYSTEMS*, MacGraw Hill
2. R. Elmasri, S.B. Navathe, *Fundamentals of database systems*, 3e édition, 1007 pages, 2000, Addison Wesley
3. Ullman J.D. and Widom J. *A First Course in Database Systems*, Prentice Hall, 1997

4. Garcia-Molina H., Ullman J. and Widom J., *Implementation of Database Systems*, Prentice Hall, 1999
5. Ullman J.D., *Principles of Database and Knowledge-Base Systems*, 2 volumes, Computer Science Press
6. Abiteboul S., Hull R., Vianu V., *Foundations of Databases*, Addison-Wesley

Le standard SQL

1. Date C.J., *A Guide to the SQL Standard*, Addison-Wesley

Trois Systèmes

1. Date C.J., *A Guide to DB2*, Addison-Wesley
2. Date C.J., *A Guide to Ingres*, Addison-Wesley
3. *ORACLE version 7 Server Concepts Manual 1992 Oracle*

Plan

Plan général du cours

1. Modèle et langages relationnels
2. Aspects systèmes des modèles relationnels
3. Concurrence et reprise sur pannes

Plan de la première partie

1. Introduction
2. Modèle relationnel
3. Algèbre Relationnelle
4. Langage de requête SQL
5. Calcul relationnel

Plan de la deuxième partie

1. Fichiers, Organisation Physique,
2. Différents tyes d'index, Arbre-B
3. Algorithmes de Jointure
4. Optimisation des requêtes;
5. l'exemple d'ORACLE.

Exemples d'Applications

1. **CLASSIQUES**
   - Gestion (salaires, stocks, . . .)
   - Transactionnel (comptes, centrales d'achat, . . .)
   - Réservations (avions, trains, . . .)
2. **PLUS RECENTES**
   - Librairie et commerce électroniques (bibliothèques, journaux, web, . . .)
   - Documentation technique (nomenclature, plans, dessins,. . .)
   - Multimédia (textes, images, son, vidéo, . . .)
   - Géographique (cartes routières, thématiques, . . .)
   - Génie Logiciel (programmes, manuels, . . .)

Comment Stocker et Manipuler les Données?

DONNÉES → BASE DE DONNÉES (B.D.)

- Une B.D. est un *GROS ENSEMBLE* d'informations *STRUCTURÉES* mémorisées sur un support *PERMANENT*.

LOGICIEL → SGBD

- Un Système de Gestion de Bases de Données (SGBD) est un logiciel de *HAUT NIVEAU* qui permet de manipuler ces informations.

Diversité -> Complexité

Diversité des utilisateurs, des interfaces et des Architectures:

1. diversité des utilisateurs: administrateurs, programmeurs,
   non informaticiens, . . .
2. diversité des interfaces: utilisateur final, langages BD, menus, saisies,
   rapports, . . .
3. diversité des architectures : centralisé, distribué,
   accès à plusieurs bases hétérogènes accessibles
   par réseau, pair à pair

FONCTIONNALITÉS d'un SGBD

Chaque niveau du SGBD réalise un certain nombre de fonctions :

NIVEAU PHYSIQUE

- Accès aux données, gestion sur mémoire secondaire (fichiers) des données, des index
- Partage de données et gestion de la concurrence d'accès
- Reprise sur pannes (fiabilité)
- Distribution des données et interopérabilité (accès aux réseaux)

NIVEAU LOGIQUE

- Définition de la structure de données : Langage de Description de Données (LDD)
- Consultation et Mise à Jour des données : Langages de Requêtes (LR) et Langage de Manipulation de Données (LMD)

Fonctionnalités du SGBD au NIVEAU EXTERNE

- Gestion des Vues
- Environnement de programmation (intégration avec un langage de programmation)
- Interfaces conviviales et Langages de 4e Génération (L4G)
- Outils d'aides (e.g. conception de schémas)
- Outils de saisie, d'impression d'états
- Débogueurs
- Passerelles (réseaux, autres SGBD,etc. . .)

En Résumé, on Veut Gérer

un GROS VOLUME D'INFORMATIONS

- Persistantes (années) et fiables (protection sur pannes)
- Partageables (utilisateurs, programmes)
- Manipulées indépendamment de leur organisation physique

Définition du schéma de données

Modèles de données

Un modèle de données est caractérisé par :

- Une structuration des objets
- Des opérations sur ces objets

Dans un SGBD, il existe plusieurs modèles plus ou moins abstraits des mêmes objets, e.g. :

- le modèle conceptuel : la description du système d'information
- le modèle logique : interface avec le SGBD
- le modèle physique : fichiers

$\Rightarrow$ ces différents modèles correspondent aux niveaux dans l'architecture d'un SGBD.

Modèle Conceptuel: Exemple Entité-Association

- Modèle très abstrait, pratique pour :
  - l'analyse du monde réel
  - la conception du système d'information
  - la communication entre différents acteurs de l'entreprise
- **Mais n'est pas associé à un langage**.

DONC UNE STRUCTURE

MAIS PAS

D'OPÉRATIONS

Modèle logique

1. **Langage de définition de données (LDD)** pour décrire la structure.
2. **Langage de manipulation de données (LMD)** pour appliquer des opérations aux données.

Ces langages sont **abstraits** :

1. Le LDD est indépendant de la représentation physique des données.
2. Le LMD est indépendant de l'implantation des opérations.

Les avantages de l'abstraction

1. **Simplicité d'accès**

   Les structures et les langages sont plus simples, donc plus faciles pour l'usager non expert.

2. **INDÉPENDANCE PHYSIQUE**.

   On peut modifier l'implantation physique sans modifier les programmes d'application

3. **INDÉPENDANCE LOGIQUE**.

   On peut modifier les programmes d'application sans toucher à l'implantation.

HISTORIQUE DES SGBD

À chaque génération correspond un modèle logique

Les premiers étaient peu abstraits (navigationnels)

< 60	S.G.F. (e.g. COBOL)	
mi-60	HIÉRARCHIQUE IMS (IBM)	navigationnel
	RÉSEAU (CODASYL)	navigationnel
73-80	RELATIONNEL	déclaratif
mi-80	RELATIONNEL	explosion sur micro
Fin 80	ORIENTÉ-OBJET	déclaratif + navigationnel
Fin 90	Objet-relationnel	nouvelles appli
2003	XML	documents

Opérations sur les données

Exemples de questions (requêtes) posées à la base

Insérer un employé nommé Jean

Augmenter Jean de 10%

Détruire Jean

Chercher les employés cadres

Chercher les employés du département comptabilité

Salaire moyen des employés comptables, avec deux enfants,

nés avant 1960 et travaillant à Paris

Les requêtes sont émises avec un *langage de requêtes* (SQL2, OQL, SQL3, XQUERY, etc.).

Le Traitement d'une Requête

- **ANALYSE SYNTAXIQUE**
- **OPTIMISATION**

  Génération (par le SGBD) d'un programme optimisé
  à partir de la
  connaissance de la structure des données, de l'existence
  d'index, de statistiques sur les données.
- **EXÉCUTION POUR OBTENIR LE RÉSULTAT**.

  NB : on
  doit tenir compte du fait que d'autres utilisateurs sont
  peut-être en train de modifier les données qu'on interroge !

Concurrence d'accès et reprise sur pannes

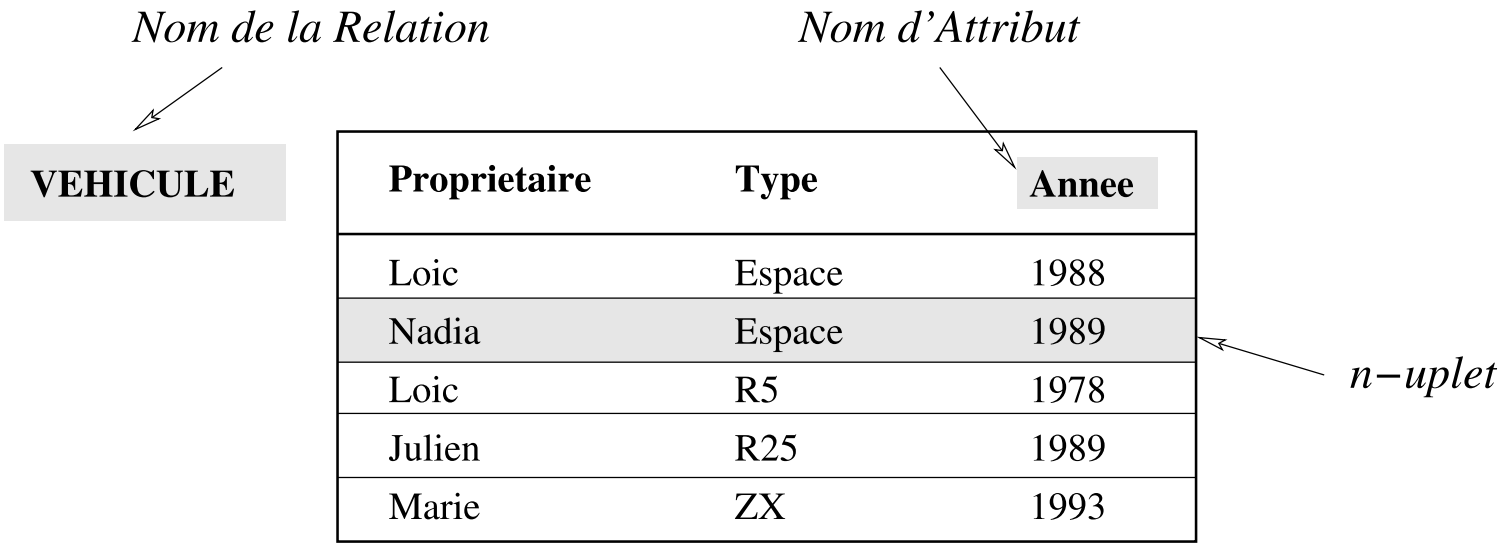
Plusieurs utilisateurs doivent pouvoir accéder en même temps aux mêmes données. Le SGBD doit savoir :

- Gérer les conflits si les deux font des mises-à-jour sur les mêmes données.
- Offrir un mécanisme de retour en arrière si on décide d'annuler des modifications en cours.
- Restaurer la base et revenir à un état cohérent en cas de panne.

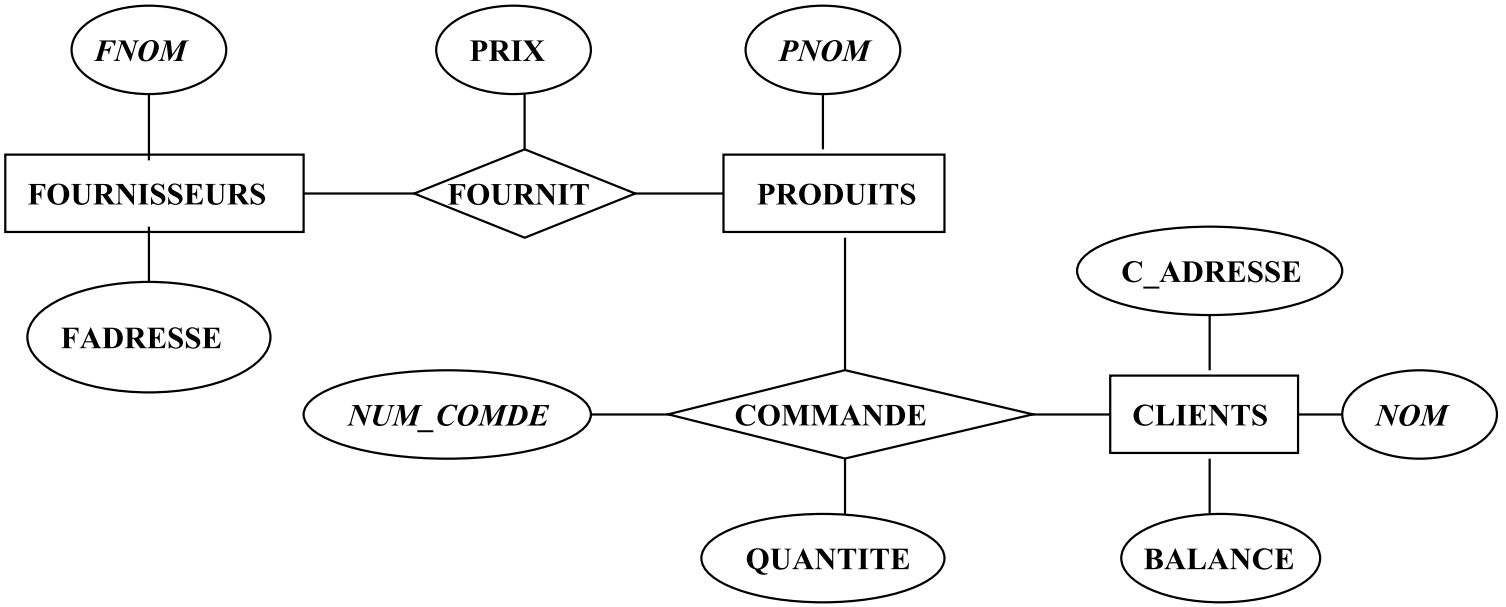
LE MODèLE RELATIONNEL

Présentation Générale

Exemple de Relation



un exemple de conception (schéma EA)



Représentation par un ensemble de relations

FOURNISSEUR

FNOM	FADRESSE
Abounayan	92190 Meudon
Cima	75010 Paris
Preblocs	92230 Gennevilliers
Sarnaco	75116 Paris

FOURNITURE

FNOM	PNOM	PRIX
Abounayan	sable	300
Abounayan	briques	1500
Preblocs	parpaing	1200
Sarnaco	parpaing	1150
Sarnaco	ciment	125

COMMANDES

NUM_COMDE	NOM	PNOM	QUANTITE
1	Jean	briques	5
2	Jean	ciment	10
3	Paul	briques	3
4	Paul	parpaing	9
5	Vincent	parpaing	7

CLIENTS

NOM	CADRESSE	BALANCE
Jean	75006 Paris	-12000
Paul	75003 Paris	0
Vincent	94200 Ivry	3000
Pierre	92400 Courbevoie	7000

Retour sur le Modèle EA

1) Entités: un fournisseur est une entité

1. Type d'entités: tous les fournisseurs ont même type: *Fournisseurs*
2. Attribut: représente une propriété (caractéristique) d'1 entité, exemple: FNOM
3. concept: synonyme d'entité.

Modèle EA

2) Associations: fournit est une association

1. relie deux (ou plus) entités, exemple: *fournit* relie Fournisseurs et Produits
2. Attribut: Une association peut avoir des attributs propres
3. role: synonyme d'association.
4. *association 1,n* entre *E* et *F*: à 1 entité de type *E* on peut associer 1 ou plusieurs entités de type *F*: exemple rajoutons l'association *siège social* entre *Fournisseurs* et *Ville*
5. *Association n,n*: à 1 entité *E* correspond 1 ou plusieurs entités *F* et réciproquement, exemple: *Founit*.

Clé

1. Attribut ou groupe d'attributs identifiant 1 entité: 2 entités de même type ne peuvent pas avoir la même valeur de clé, exemple *FNOM* pour Fournisseurs (hyp: 2 forunisseurs ne peuvent avoir le même nom)
2. *Clé primaire*: il peut y avoir plusieurs clés, exemple : si on rajoute le no de SS comme attribut à Fournisseurs, on a deux clés. Une est choisie comme clé primaire, e.g; NoSS

Passage EA-Relationnel

A partir d'un schéma entité-association (niveau conceptuel) comment passer au niveau logique (comment choisir les relations)?

1. Une relation par type d'entité (e.g; Fournisseur). A une entité correspond un nuplet.
2. la clé primaire de la relation (attribut(s) de la relation qui identifie un nuplet) est celle du type d'entité
3. Une relation par association n,n: Les attributs sont les clés primaires des relations reliées par l'association, ainsi que les attributs propres de l'association
4. *Clé étrangère*: si l'association A entre R et S est 1,n, (n entités de type S pour une entité de type R), on ne crée pas de relation associée à cette association: on rajoute dans la table qui représente S la clé de R comme attribut. Celle-ci et appelée clé étrangère.

5. exercice: rajouter le type d'entité *VILLE* (clé: nom de ville) et l'association *siège social* entre *VILLE* et *FOURNISSEURS*. Comment est modifiée la relation *FOURNISSEURS*?

Modèle relationnel: Définitions

Définitions

- Un Domaine est un ensemble de valeurs. Exemples : $\{0, 1\}$, N, l'ensemble des chaînes de caractères, l'ensemble des chaînes de caractères de longueur 10.
- Un ATTRIBUT prend ses valeurs dans un domaine. Plusieurs attributs peuvent avoir le même domaine.
- Un NUPLET est une liste de n valeurs $(v_1, \ldots, v_n)$ où chaque valeur v_i est la valeur d'un attribut A_i de domaine D_i : $v_i \in D_i$
- Le PRODUIT CARTÉSIEN $D_1 \times \ldots \times D_n$ entre des domaines $D_1, \ldots, D_n$ est l'ensemble de **tous** les nuplets $(v_1, \ldots, v_n)$ où $v_i \in D_i$.

- RELATION : soit $D_1, \ldots, D_n$ les domaines respectifs des attributs $A_1, \ldots, A_n$. Une relation R définie sur les attributs $A_1, \ldots, A_n$ est un sous-ensemble fini du produit cartésien $D_1 \times \ldots \times D_n$: R est un ensemble de nuplets.
- Une relation R est représentée sous forme d'une **table**. L'ordre des colonnes ou des lignes n'a pas d'importance. Les colonnes sont distinguées par les noms d'attributs et chaque ligne représente un élément de l'ensemble R (un nuplet).
- Un attribut peut apparaître dans plusieurs relations.
- Une BASE DE DONNÉES est un ensemble de relations.

- L'UNIVERS D'ATTRIBUTS D'UNE BASE DE DONNÉES est l'ensemble de tous les attributs des relations de la base.
- Le SCHÉMA D'UNE RELATION R est défini par le nom de la relation et la liste des attributs avec pour chaque attribut son domaine.
  Notation :

$$R(A_1 : D_1, \ldots, A_n : D_n)$$

  ou plus simplement :

$$R(A_1, \ldots, A_n)$$

  Exemple :

  VEHICULE(NOM:CHAR(20), TYPE:CHAR(10),
  ANNEE:ENTIER)

- Si la relation a n attributs (n colonnes), n est appelé ARITÉ de la relation. La relation $VEHICULE$ est d'arité 3.
- Le SCHÉMA D'UNE BASE DE DONNÉES est l'ensemble des schémas de ses relations.

Exemple de Base de Données

SCHÉMA :

- FOURNISSEURS(FNOM:CHAR(20), FADRESSE:CHAR(30))
- FOURNITURE(FNOM:CHAR(20), PNOM:CHAR(10), PRIX:ENTIER))
- COMMANDES(NUM_COMDE:ENTIER, NOM:CHAR(20), PNOM:CHAR(10), QUANTITE;ENTIER))
- CLIENTS(NOM: CHAR(20), CADRESSE:CHAR(30), BALANCE:RELATIF)

Exemple de Base de Données

UNIVERS D'ATTRIBUTS :

- U={FNOM, PNOM, NOM, FADRESSE, CADRESSE,
  PRIX, NUM_CODE, QUANTITE, BALANCE}

RELATION UNIVERSELLE :

- FPCC(FNOM, PNOM, NOM, FADRESSE, CADRESSE,
  PRIX, NUM_CODE, QUANTITE, BALANCE)

Opérations et Langages

Opérations sur une Base de Données Relationnelle

- LANGAGE DE DÉFINITION DES DONNÉES (définition et MAJ du schéma) :
  - Création et destruction d'une relation ou d'une base
  - Ajout, suppression d'un attribut

- LANGAGE DE MANIPULATION DES DONNÉES
  - Saisie des nuplets d'une relation
  - Affichage d'une relation
  - Modification d'une relation : insertion, suppression et maj des nuplets
  - Requêtes : consultation d'une relation ou calcul d'une nouvelle relation
- GESTION DES TRANSACTIONS
- GESTION DES VUES

Langages de Requêtes Relationnels

POUVOIR D'EXPRESSION : Qu'est-ce qu'on peut calculer ? Quelles opérations peut-on faire ?

Les langages de requête relationnels utilisent deux approches :

- calcul relationnel
- algèbre relationnelle

Les deux approches ont **même pouvoir d'expression**.

ALGÈBRE RELATIONNELLE

Algèbre Relationnelle

Opérations relationnels :

- une opération prend en entrée une ou deux relations
- le résultat est toujours une relation

5 Opérations de base (pour exprimer toutes les requêtes) :

- Opérations unaires : sélection, projection
- Opérations binaires : union, différence, produit cartésien
- Autres opérations qui s'expriment en fonction des 5 opérations de base : jointure (naturelle, θ-jointure), intersection, division

Projection

LA PROJECTION “ÉLIMINE” UNE OU PLUSIEURS COLONNES D’UNE RELATION.

Notation :

$$\pi A_1, A_2, \ldots, A_k (R)$$

Projection: Exemples

a) On élimine la colonne C dans la relation R :

R	A	B	C
$\rightarrow$	a	b	c
	d	a	b
	c	b	d
$\rightarrow$	a	b	e
	e	e	a

$\Rightarrow$ $\pi_{A,B}(R)$

A	B
a	b
d	a
c	b
e	e

Le nuplet (a, b) n'apparaît qu'**une** fois dans la relation $\pi_{A,B}(R)$, bien qu'il existe **deux** nuplets (a, b, c) et (a, b, e) dans R.

Projection: Exemples

b) On élimine la colonne B dans la relation R (on garde A et C) :

R

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

$\Rightarrow$

$\pi_{A,C}(R)$

A	C
a	c
d	b
c	d
a	e
e	a

Sélection

Sélection sur la condition $\mathcal{C}$:

On garde les nuplets qui satisfont $\mathcal{C}$.

NOTATION :

$$\sigma_C(R)$$

Sélection: Exemples

a) On sélectionne les nuplets dans la relation R tels que l'attribut B vaut "b" :

R

A	B	C
a	b	1
d	a	2
c	b	3
a	b	4
e	e	5

$\Rightarrow$ $\sigma_{B=''b''}(R)$

A	B	C
a	b	1
c	b	3
a	b	4

Sélection: Exemples

b) On sélectionne les nuplets tels que

$$(A ='' a'' \vee B ='' a'') \wedge C \leq 3 :$$

R

A	B	C
a	b	1
d	a	2
c	b	3
a	b	4
e	e	5

$\Rightarrow \quad \sigma_{(A=''a'' \vee B=''a'') \wedge C \leq 3}(R)$

A	B	C
a	b	1
d	a	2

Sélection: Exemples

c) On sélectionne les nuplets tels que la 1re et la 2e colonne sont identiques :

R

A	B	C
a	b	1
d	a	2
c	b	3
a	b	4
e	e	5

$\Rightarrow \quad \sigma_{A=B}(R)$

A	B	C
e	e	5

Condition de Sélection

La condition $\mathcal{C}$ d'une sélection peut être une **formule logique** quelconque avec des **et ($\wedge$)** et des **ou ($\vee$)** entre termes de la forme $A_i \theta A_j$ et $A_i \theta a$ où

- A_i et A_j sont des attributs,
- a est un élément (une valeur) du domaine de A_i,
- θ est l'un de $=, <, \leq, >, \geq, \neq$.

Expressions de l'Algèbre Relationnelle

- le résultat d'une opération est une **relation**
- sur cette relation, on peut faire une **autre opération** de l'algèbre

$\Rightarrow$ *Les opérations peuvent être* composées *pour former des expressions de l'algèbre relationnelle.*

Expressions de l'Algèbre Relationnelle

EXEMPLE : $COMMANDES(NOM, PNOM, NUM, QTE)$

$$R'' = \pi_{PNOM}(\overbrace{\sigma_{NOM=''Jean''}(COMMANDES)}^{R'})$$

La relation $R'(NOM, PNOM, NUM, QTE)$ contient les nuplets dont l'attribut NOM a la valeur $''Jean''$. La relation $R''(PNOM)$ contient tous les produits commandés par Jean.

Produit Cartésien

- NOTATION : $R \times S$
- ARGUMENTS : 2 relations quelconques :

$$R(A_1, A_2, \ldots, A_n) \;\; S(B_1, B_2, \ldots, B_k)$$

- SCHÉMA DE $T = R \times S$: $T(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_k)$
- VALEUR DE $T = R \times S$: ensemble de tous les nuplets ayant $n + k$ composants (attributs)
  - dont les n premiers composants forment un nuplet de R
  - et les k derniers composants forment un nuplet de S

Exemple de Produit Cartésien

R

A	**B**
1	1
1	2
3	4

$|R|$

S

C	**D**	**E**
a	b	a
a	b	c
b	a	a

$|S|$

$\Rightarrow$

$\mathbf{R} \times \mathbf{S}$

$| R | \times | S |$

A	**B**	**C**	**D**	**E**
1	1	a	b	a
1	1	a	b	c
1	1	b	a	a
1	2	a	b	a
1	2	a	b	c
1	2	b	a	a
3	4	a	b	a
3	4	a	b	c
3	4	b	a	a

Jointure Naturelle

- NOTATION : $R \bowtie S$
- ARGUMENTS : 2 relations quelconques :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k) \;\; S(B_1, \ldots, B_n, X_1, \ldots, X_k)$$

où $X_1, \ldots, X_k$ sont les attributs en commun.

- SCHÉMA DE $T = R \bowtie S$: $T(A_1, \ldots, A_m, B_1, \ldots, B_n, X_1, \ldots, X_k)$
- VALEUR DE $T = R \bowtie S$: ensemble de tous les nuplets ayant $m + n + k$ attributs dont les m premiers et k derniers composants forment un nuplet de R et les $n + k$ derniers composants forment un nuplet de S.

Jointure Naturelle: Exemple

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

⇒

R ⋈ S

A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

Jointure Naturelle

Soit $U = \{A_1, \ldots, A_m, B_1, \ldots, B_n, X_1, \ldots, X_k\}$ l'ensemble des attributs des 2 relations et $V = \{X_1, \ldots, X_k\}$ l'ensemble des attributs en commun.

$$R \bowtie S = \pi_U(\sigma_{\forall A \in V : R.A = S.A}(R \times S))$$

NOTATION : $R.A$ veut dire "l'attribut A de la relation R".

Jointure Naturelle: Exemple

R

A	B
1	a
1	b
4	a

S

A	B	D
1	a	b
2	c	b
4	a	a

$\Rightarrow$

R × S	**R.A**	**R.B**	**S.A**	**S.B**	**D**
	1	a	1	a	b
→	**1**	**a**	**2**	**c**	b
→	**1**	**a**	**4**	**a**	a
→	**1**	**b**	**1**	**a**	b
→	**1**	**b**	**2**	**c**	b
→	**1**	**b**	**4**	**a**	a
→	**4**	**a**	**1**	**a**	b
→	**4**	**a**	**2**	**c**	b
	4	a	4	a	a

⇓

R ⋈ S

A	**B**	**D**
1	a	b
4	a	a

$$\Leftarrow \qquad \pi_{R.A,R.B,D}\big(\sigma_{R.A=S.A \wedge R.B=S.B}(R \times S)\big)$$

Jointure Naturelle: Algorithme

Pour chaque nuplet a dans R et pour chaque nuplet b dans S :

1. on concatène a et b et on obtient un nuplet qui a pour attributs

$$\overbrace{A_1, \ldots, A_m, X_1, \ldots, X_k}^{a}, \overbrace{B_1, \ldots, B_n, X_1, \ldots, X_k}^{b}$$

2. on ne le garde que si chaque attribut X_i de a est égal à l'attribut X_i de b : $\forall_{i=1..k} a.X_i = b.X_i$.

3. on élimine les valeurs (colonnes) dupliquées : on obtient un nuplet qui a pour attributs

$$\overbrace{A_1, \ldots, A_m}^{a}, \overbrace{B_1, \ldots, B_m}^{b}, \overbrace{X_1, \ldots, X_k}^{a\ et\ b}$$

θ-Jointure

- ARGUMENTS : 2 relations quelconques :

$$R(A_1, \ldots, A_m) \qquad S(B_1, \ldots, B_n)$$

- NOTATION : $R \bowtie_{A_i \theta B_j} S, \theta \in \{=, \neq, <, \leq, >, \geq\}$
- SCHÉMA DE $T = R \bowtie_{A_I \theta B_J} S$: $T(A_1, \ldots, A_m, B_1, \ldots, B_n)$
- VALEUR DE $T = R \bowtie_{A_I \theta B_J} S$: $T = \sigma_{A_i \theta B_j}(R \times S))$
- ÉQUIJOINTURE : θ est l'**égalité**.

θ-Jointure: Exemple

R

A	B
1	a
1	b
3	a

S

C	D	E
1	b	a
2	b	c
4	a	a

T := R × S	**A**	**B**	**C**	**D**	**E**
	1	a	1	b	a
	1	a	2	b	c
	1	a	4	a	a
	1	b	1	b	a
	1	b	2	b	c
	1	b	4	a	a
$A > C \rightarrow$	**3**	a	**1**	b	a
$A > C \rightarrow$	**3**	a	**2**	b	c
	3	a	4	a	a

$\Rightarrow$

$\sigma_{A \leq C}(\mathbf{T})$
$= \mathbf{R} \bowtie_{A \leq C} \mathbf{S}$

A	B	C	D	E
1	a	1	b	a
1	a	2	b	c
1	a	4	a	a
1	b	1	b	a
1	b	2	b	c
1	b	4	a	a
3	a	4	a	a

Équijointure: Exemple

R

A	B
1	a
1	b
3	a

S

C	D	E
1	b	a
2	b	c
4	a	a

T := R × S	**A**	**B**	**C**	**D**	**E**
$B \neq D \rightarrow$	1	**a**	1	**b**	a
$B \neq D \rightarrow$	1	**a**	2	**b**	c
	1	a	4	a	a
	1	b	1	b	a
	1	b	2	b	c
$B \neq D \rightarrow$	1	**b**	4	**a**	a
$B \neq D \rightarrow$	3	**a**	1	**b**	a
$B \neq D \rightarrow$	3	**a**	2	**b**	c
	3	a	4	a	a

$\Rightarrow$

$\sigma_{B=D}$**(T)**
=**R** $\bowtie_{B=D}$ **S**

A	**B**	**C**	**D**	**E**
1	a	4	a	a
1	b	1	b	a
1	b	2	b	c
3	a	4	a	a

Équijointure vs. Jointure Naturelle

$$IMMEUBLE(ADI, NBETAGES, DATEC, PROP)$$

$$APPIM(ADI, NAP, OCCUP, ETAGE)$$

1. Nom du propriétaire de l'immeuble où est situé l'appartement occupé par *Durand* :

$$\pi_{PROP}(\overbrace{IMMEUBLE \bowtie \sigma_{OCCUP=''DURAND''}(APPIM)}^{JointureNaturelle})$$

2. Appartements occupés par des propriétaires d'immeuble :

$$\pi_{ADI,NAP,ETAGE}\overbrace{(APPIM \bowtie_{OCCUP=PROP} IMMEUBLE)}^{équijointure})$$

Autre Exemple de REQUÊTE : Nom et adresse des clients qui ont commandé des parpaings:

- Schéma Relationnel :

$$COMMANDES(PNOM, CNOM, NUM\_CMDE, QTE)$$

$$CLIENTS(CNOM, CADRESSE, BALANCE)$$

- Requête Relationnelle :

$$\pi_{CNOM,CADRESSE}(CLIENTS \bowtie \sigma_{PNOM=''PARPAING''}(COMMANDES))$$

Union

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \quad S(A_1, \ldots, A_m)$$

- NOTATION : $R \cup S$
- SCHÉMA DE $T = R \cup S$: $T(A_1, \ldots, A_m)$
- VALEUR DE T : Union ensembliste sur $D_1 \times \ldots \times D_m$:

$$T = \{t \mid t \in R \vee t \in S\}$$

Union: Exemple

R

A	B
a	b
a	c
d	e

S

A	B
a	b
a	e
d	e
f	g

$\Rightarrow$

R ∪ S

A	B
a	b
a	c
d	e
a	e
f	g

→ (a, b)

→ (d, e)

Différence

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \quad S(A_1, \ldots, A_m)$$

- NOTATION : $R - S$
- SCHÉMA DE $T = R - S$: $T(A_1, \ldots, A_m)$
- VALEUR DE T : Différence ensembliste sur $D_1 \times \ldots \times D_m$:

$$T = \{t \mid t \in R \wedge t \notin S\}$$

Différence: Exemple

R	A	B
→	a	b
	a	c
→	d	e

S	A	B
→	a	b
	a	e
→	d	e
	f	g

R - S	A	B
	a	c

S - R	A	B
	a	e
	f	g

Intersection

- ARGUMENTS : 2 relations de même schéma :

$$R(A_1, \ldots, A_m) \quad S(A_1, \ldots, A_m)$$

- NOTATION : $R \cap S$
- SCHÉMA DE $T = R \cap S$: $T(A_1, \ldots, A_m)$
- VALEUR DE T : Intersection ensembliste sur $D_1 \times \ldots \times D_m$:

$$T = \{t \mid t \in R \wedge t \in S\}$$

Intersection: Exemple

R	A	B
→	a	b
	a	c
→	d	e

S	A	B
→	a	b
	a	e
→	d	e
	f	g

R - S

A	B
a	c

R ∩ S = R - (R - S)

A	B
a	b
d	e

Semijointure

- ARGUMENTS : 2 relations quelconques :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k) S(B_1, \ldots, B_n, X_1, \ldots, X_k)$$

où $X_1, \ldots, X_k$ sont les attributs en commun.

- NOTATION : $R \ltimes S$
- SCHÉMA DE $T = R \ltimes S$: $T(A_1, \ldots, A_m, X_1, \ldots, X_k)$
- VALEUR DE $T = R \ltimes S$: Projection sur les attributs de R de la jointure naturelle entre R et S.

Semijointure

La semijointure correspond à une sélection où la condition de sélection est définie par le biais d'une autre relation.

Soit $U = \{A_1, \ldots, A_m\}$ l'ensemble des attributs de R.

$$R \ltimes S = \pi_U(R \bowtie S)$$

Semijointure: Exemple

R

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

S

B	C	D
b	c	d
b	c	e
a	d	b

$\Rightarrow \pi_{A,B,C}(R \bowtie S) \Rightarrow$

R $\rtimes$ **S**

A	B	C
a	b	c
d	b	c
c	a	d

Division: Exemple

REQUÊTE : Clients qui commandent tous les produits:

COMM

NUM	NOM	PNOM	QTE
1	Jean	briques	100
2	Jean	ciment	2
3	Jean	parpaing	2
4	Paul	briques	200
5	Paul	parpaing	3
6	Vincent	parpaing	3

$R = \pi_{NOM,PNOM}(COMM)$:

R

NOM	PNOM
Jean	briques
Jean	ciment
Jean	parpaing
Paul	briques
Paul	parpaing
Vincent	parpaing

PROD

PNOM
briques
ciment
parpaing

$\Downarrow$

$R \div PROD$

NOM
Jean

Division: Exemple

R

A	B	C	D
a	b	x	m
a	b	y	n
a	b	z	o
b	c	x	o
b	d	x	m
c	e	x	m
c	e	y	n
c	e	z	o
d	a	z	p
d	a	y	m

S

C	D
x	m
y	n
z	o

R÷S

A	B
a	b
c	e

Division

- ARGUMENTS : 2 relations :

$$R(A_1, \ldots, A_m, X_1, \ldots, X_k) \quad S(X_1, \ldots, X_k)$$

où **tous** les attributs de S sont des attributs de R.

- NOTATION : $R \div S$
- SCHÉMA DE $T = R \div S$: $T(A_1, \ldots, A_m)$
- VALEUR DE $T = R \div S$:

$$R \div S = \{(a_1, \ldots, a_m) \mid \forall (x_1, \ldots, x_k) \in S : (a_1, \ldots, a_m, x_1, \ldots, x_k) \in R\}$$

Division

La division s'exprime en fonction du produit cartésien, de la projection et de la différence : $R \div S = R_1 - R_2$ où

$$R_1 = \pi_{A_1,...,A_m}(R) \; et \; R_2 = \pi_{A_1,...,A_m}((R_1 \times S) - R)$$

Renommage

- NOTATION : ρ
- ARGUMENTS : 1 relation :

$$R(A_1, \ldots, A_n)$$

- SCHÉMA DE $T = \rho_{A_i \to B_i} R$: $T(A_1, \ldots, A_{i-1}, B_i, A_{i+1}, \ldots, A_n)$
- VALEUR DE $T = \rho_{A_i \to B_i} R$: $T = R$. La valeur de R est inchangée.
  Seul le nom de l'attribut A_i a été remplacé par B_i

SQL

Principe

- SQL (Structured Query Language) est le Langage de Requêtes standard pour les SGBD relationnels
- Expression d'une requête par un bloc *SELECT FROM WHERE*

  SELECT <liste des attributs a projeter>
  FROM <liste des relations arguments>
  WHERE <conditions sur un ou plusieurs attributs>

- Dans les requêtes simples, la correspondance avec l'algèbre relationnelle est facile à mettre en évidence.

Historique*

SQL86 - SQL89 ou SQL1 La référence de base:

- Requêtes compilées puis exécutées depuis un programme d'application.
- Types de données simples (entiers, réels, chaînes de caractères de taille fixe)
- Opérations ensemblistes restreintes (UNION).

SQL92 ou SQL2 Standard actuel:

- Requêtes dynamiques: exécution différée ou immédiate
- Types de données plus riches (intervalles, dates, chaînes de caractères de taille variable)
- Différents types de jointures: jointure naturelle, jointure externe
- Opérations ensemblistes: différence (EXCEPT), intersection

(INTERSECT)

- Renommage des attributs dans la clause SELECT

SQL3 (en cours) : SQL devient un langage de programmation :

- Extensions orientées-objet
- Opérateur de fermeture transitive (recursion)
- ...

Expressions de Base

Projection

Soit le schéma de relation
COMMANDES(NUM,CNOM,PNOM,QUANTITE)

REQUÊTE: *Information sur toutes les commandes*

SQL:

```
SELECT NUM,CNOM,PNOM,QUANTITE
FROM   COMMANDES
```

ou

```
SELECT *
FROM   COMMANDES
```

Projection : Distinct

Soit le schéma de relation
COMMANDES(NUM,CNOM,PNOM,QUANTITE)

REQUÊTE: *Produits commandés*

```
SELECT PNOM
FROM   COMMANDES
```

NOTE: Contrairement à l'algèbre relationnelle, SQL n'élimine pas les dupliqués. Pour les éliminer on utilise DISTINCT :

```
SELECT DISTINCT PNOM
FROM   COMMANDES
```

Le DISTINCT peut être remplacé par la clause UNIQUE dans certains systèmes

Sélection

Soit le schéma de relation
COMMANDES(NUM,CNOM,PNOM,QUANTITE)

REQUÊTE: *Produits commandés par Jean*

ALGÈBRE: $\pi_{PNOM}(\sigma_{CNOM="JEAN"}(COMMANDES))$

SQL:

```
SELECT  PNOM
FROM    COMMANDES
WHERE   CNOM = 'JEAN'
```

REQUÊTE: *Produits commandés par Jean en quantité supérieure à 100*

SQL:

```
SELECT  PNOM
FROM    COMMANDES
WHERE   CNOM = 'JEAN'
AND     QUANTITE > 100
```

Conditions de sélection en SQL : Conditions simples

Les conditions de base sont exprimées de deux façons:

1. *attribut comparateur valeur*
2. *attribut comparateur attribut*

où $comparateur$ est $=, <, >, <>, \ldots,$

Soit le schéma de relation **FOURNITURE**(PNOM,FNOM,PRIX)

REQUÊTE: *Produits de prix supérieur à 200F*

SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX > 2000
```

Soit le schéma de relation **FOURNITURE**(PNOM,FNOM,PRIX)

REQUÊTE: *Produits dont le nom est celui du fournisseur*

SQL:

```
SELECT  PNOM
FROM    FOURNITURE
WHERE   PNOM = FNOM
```

Conditions de sélection en SQL : Suite

Le *comparateur* est BETWEEN, LIKE, IN

Soit le schéma de relation **FOURNITURE**(PNOM,FNOM,PRIX)

REQUÊTE: *Produits avec un coût entre 1000F et 2000F*

SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX BETWEEN 1000 AND 2000
```

NOTE: La condition y BETWEEN x AND z est équivalente à $y <= z$
AND
$x <= y$.

Soit le schéma de relation
COMMANDES(NUM,CNOM,PNOM,QUANTITE)

REQUÊTE: *Clients dont le nom commence par "C"*

SQL:

```
SELECT CNOM
FROM   COMMANDES
WHERE  CNOM LIKE 'C%'
```

NOTE: Le littéral qui suit LIKE doit être une chaîne de caractères éventuellement avec des caractères jokers (_, %). Pas exprimable avec l'algèbre relationnelle.

Soit le schéma de relation **FOURNITURE**(PNOM,FNOM,PRIX)

REQUÊTE: *Produits avec un coût de 100F, de 200F ou de 300F*

SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX IN {100,200,300}
```

NOTE: La condition x IN $\{a, b, \ldots, z)$ est équivalente à $x = a$ OR $x = b$ OR ... OR $x = z$.

Jointure

Soit le schéma de relations

COMMANDES(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)

REQUÊTE: *Nom, Coût, Fournisseur des Produits commandés par Jean*

ALGÈBRE :

$$\pi_{PNOM,PRIX,FNOM}(\sigma_{CNOM=\text{"}JEAN\text{"}}(COMMANDES) \bowtie (FOURNITURE))$$

SQL :

```
SELECT COMMANDES.PNOM, PRIX, FNOM
FROM   COMMANDES, FOURNITURE
WHERE  CNOM = 'JEAN' AND
       COMMANDES.PNOM = FOURNITURE.PNOM
```

NOTE: Cette requête est équivalente à une jointure naturelle. Noter qu'il faut toujours expliciter les attributs de jointure.

NOTE: SELECT COMMANDES.PNOM, PRIX, FNOM FROM COMMANDES, FOURNITURE équivaut à un produit cartésien des deux relations, suivi d'une projection.

Soit le schéma de relation **FOURNISSEUR**(FNOM,STATUT,VILLE)

REQUÊTE: *Fournisseurs qui habitent deux à deux dans la même ville*

SQL:

```
SELECT PREM.FNOM, SECOND.FNOM
FROM   FOURNISSEUR PREM, FOURNISSEUR SECO
WHERE  PREM.VILLE =  SECOND.VILLE AND
       PREM.FNOM < SECOND.FNOM
```

La deuxième condition permet

1. l'élimination des paires (x,x)
2. d'éviter d'obtenir au résultat à la fois (x,y) et (y,x)

NOTE: PREM représente une instance de FOURNISSEUR, SECOND une autre instance de FOURNISSEUR.

Soit le schéma de relation **EMPLOYE**(EMPNO,ENOM,DEPNO,SAL)

REQUÊTE: *Nom et Salaire des Employés gagnant plus que l'employé de numéro 12546*

ALGÈBRE:

$$\text{R1} := \pi_{SAL}(\sigma_{EMPNO=12546}(EMPLOYE))$$

$$\text{R2} := \pi_{ENOM,EMPLOYE.SAL}((EMPLOYE) \bowtie_{EMPLOYE.SAL>R1.SAL} (R1))$$

SQL:

```
SELECT E1.ENOM, E1.SAL
FROM   EMPLOYE E1, EMPLOYE E2
WHERE  E2.EMPNO = 12546 AND
       E1.SAL > E2.SAL
```

Expressions Ensemblistes

Union

COMMANDES(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)

REQUÊTE: *Produits qui coûtent plus que 1000F ou ceux qui sont commandés par Jean*

ALGÈBRE:

$$\pi_{PNOM}(\sigma_{PRIX>1000}(FOURNITURE))$$

$$\cup$$

$$\pi_{PNOM}(\sigma_{CNOM='Jean'}(COMMANDES))$$

SQL:

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX >= 1000
UNION
SELECT PNOM
FROM   COMMANDES
WHERE  CNOM = 'Jean'
```

NOTE: L'union élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `UNION ALL` : le résultat contient chaque n-uplet $a + b$ fois, où a et b sont le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

Différence

La différence ne fait pas partie du standard.

EMPLOYE(EMPNO,ENOM,DEPTNO,SAL)
DEPARTEMENT(DEPTNO,DNOM,LOC)

REQUÊTE: *Départements sans employés*

ALGÈBRE:
$\pi_{DEPTNO}(DEPARTEMENT) - \pi_{DEPTNO}(EMPLOYE)$

SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
EXCEPT
SELECT DEPTNO
FROM   EMPLOYE
```

NOTE: La différence élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `EXCEPT ALL` :

Intersection

L'intersection ne fait pas partie du standard.

EMPLOYE(EMPNO,ENOM,DEPTNO,SAL)
DEPARTEMENT(DEPTNO,DNOM,LOC)

REQUÊTE: *Départements ayant des employés qui gagnent plus que 20000F et qui se trouvent à Paris*

ALGÈBRE :

$$\pi_{DEPTNO}(\sigma_{LOC=\text{"}Paris\text{"}}(DEPARTEMENT))$$
$$\cap$$
$$\pi_{DEPTNO}(\sigma_{SAL>20000}(EMPLOYE))$$

SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
WHERE  LOC = 'Paris'
INTERSECT
SELECT DEPTNO
FROM   EMPLOYE
WHERE  SAL > 20000
```

NOTE: L'intersection élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `INTERSECT ALL` : le résultat contient chaque n-uplet $min(a, b)$ fois, où a et b sont le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

Imbrication des Requêtes en SQL

Requêtes imbriquées simples

La Jointure s'exprime par deux blocs SFW imbriqués

Soit le schéma de relations

COMMANDES(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)

REQUÊTE: *Nom, prix et fournisseurs des Produits commandés par Jean*

ALGÈBRE:

$$\pi_{PNOM,PRIX,FNOM}(\sigma_{CNOM="JEAN"}(COMMANDES) \bowtie (FOURNITURE))$$

SQL:

```
SELECT PNOM,PRIX,FNOM
FROM   FOURNITURE
WHERE  PNOM IN (SELECT PNOM
                FROM   COMMANDES
                WHERE  CNOM = 'JEAN')
```

ou

```
SELECT FOURNITURE.PNOM,PRIX,FNOM
FROM   FOURNITURE,COMMANDES
WHERE  FOURNITURE.PNOM = COMMANDES.PNOM
AND    CNOM = ''JEAN''
```

La Différence s'exprime aussi par deux blocs SFW imbriqués

Soit le schéma de relations

EMPLOYE(EMPNO,ENOM,DEPNO,SAL)
DEPARTEMENT(DEPTNO,DNOM,LOC)

REQUÊTE: *Départements sans employés*

ALGÈBRE :

$$\pi_{DEPTNO}(DEPARTEMENT) - \pi_{DEPTNO}(EMPLOYE)$$

SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
WHERE  DETPNO NOT IN (SELECT DISTINCT DEPTNO
                      FROM   EMPLOYE)
```

ou

```
SELECT DEPTNO
FROM DEPARTEMENT
EXCEPT
SELECT DISTINCT DEPTNO
FROM EMPLOYE
```

Requêtes imbriquées plus complexes : ANY - ALL

Soit le schéma de relation **FOURNITURE**(PNOM,FNOM,PRIX)

REQUÊTE: *Fournisseurs des Briques à un coût inférieur au coût maximum des Ardoises*

```
SQL :     SELECT FNOM
          FROM   FOURNITURE
          WHERE  PNOM = 'Brique'
          AND    PRIX < ANY (SELECT PRIX
                             FROM   FOURNITURE
                             WHERE  PNOM = 'Ardoise')
```

NOTE: La condition $f\ \theta$ ANY (SELECT F FROM ...) est vraie ssi la comparaison $f\theta v$ est vraie au moins pour une valeur v du résultat du bloc (SELECT F FROM ...).

Soit le schéma de relations

COMMANDE(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)

REQUÊTE: *Nom, Coût et Fournisseur des Produits commandés par Jean*

SQL:

```
SELECT PNOM, PRIX, FNOM
FROM   FOURNITURE
WHERE  PNOM = ANY (SELECT PNOM
                   FROM   COMMANDE
                   WHERE  CNOM = 'JEAN')
```

NOTE: Les prédicats IN et = ANY sont utilisés de façon équivalente.

Soit le schéma de relation
COMMANDE(NUM,CNOM,PNOM,QUANTITE)

REQUÊTE: *Client ayant commandé la plus petite quantité de Briques*

SQL:

```
SELECT CNOM
FROM   COMMANDE
WHERE  PNOM = 'Brique' AND
       QUANTITE <= ALL (SELECT QUANTITE
                        FROM   COMMANDE
                        WHERE  PNOM = 'Brique')
```

NOTE: La condition $f\ \theta$ ALL (SELECT F FROM ...) est vraie ssi la comparaison $f\theta v$ est vraie pour toutes les valeurs v du résultat du bloc (SELECT F FROM ...).

Soit le schéma de relations

EMPLOYE(EMPNO,ENOM,DEPNO,SAL)
DEPARTEMENT(DEPTNO,DNOM,LOC)

REQUÊTE: *Départements sans employés*

SQL:

```
SELECT DEPTNO
FROM   DEPARTEMENT
WHERE  DETPNO NOT = ALL (SELECT DISTINCT DEPTNO
                         FROM   EMPLOYE)
```

NOTE: Les prédicats NOT IN et NOT = ALL sont utilisés de façon équivalente.

Requêtes imbriquées plus complexes : EXISTS

Soit le schéma de relations

FOURNISSEUR(FNOM,STATUS,VILLE)
FOURNITURE(PNOM,FNOM,PRIX)

REQUÊTE: *Fournisseurs qui fournissent au moins un produit*

```
SQL :       SELECT FNOM
            FROM   FOURNISSEUR
            WHERE  EXISTS (SELECT *
                           FROM   FOURNITURE
                           WHERE  FNOM = FOURNISSEUR.FNOM)
```

NOTE: La condition EXISTS (SELECT * FROM . . .) est vraie ssi le résultat du bloc (SELECT F FROM . . .) n'est pas vide.

Soit le schéma de relations

FOURNISSEUR(FNOM,STATUS,VILLE)
FOURNITURE(PNOM,FNOM,PRIX)

REQUÊTE: *Fournisseurs qui ne fournissent aucun produit*

SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  NOT EXISTS (SELECT *
                   FROM   FOURNITURE
                   WHERE  FNOM = FOURNISSEUR.FNOM)
```

NOTE: La condition NOT EXISTS (SELECT * FROM . . .) est vraie ssi le résultat du bloc (SELECT F FROM . . .) est vide.

Formes Équivalentes de Quantification

Si θ est un des opérateurs de comparaison $<, =, >, \ldots$

- La condition x θ ANY (SELECT Ri.y FROM R1, . . . Rn WHERE p) est équivalente à

  EXISTS (SELECT * FROM R1, . . . Rn WHERE p AND x θ Ri.y)

- La condition x θ ALL (SELECT Ri.y FROM R1, . . . Rn WHERE p) est équivalente à

  NOT EXISTS (SELECT * FROM R1, . . . Rn WHERE (p) AND NOT (x θ Ri.y))

Soit le schéma de relations

COMMANDE(NUM,CNOM,PNOM,QUANTITE)
FOURNITURE(PNOM,FNOM,PRIX)

REQUÊTE: *Nom, prix et fournisseur des produits commandés par Jean*

```
SELECT PNOM, PRIX, FNOM  FROM   FOURNITURE
WHERE  EXISTS  (SELECT * FROM   COMMANDE
                WHERE CNOM = 'JEAN'
                  AND PNOM = FOURNITURE.PNOM)

SELECT PNOM, PRIX, FNOM FROM   FOURNITURE
WHERE  PNOM = ANY (SELECT PNOM  FROM   COMMANDE
                   WHERE  CNOM = 'JEAN')
```

Soit le schéma de relation **FOURNITURE**(PNOM,FNOM,PRIX)

REQUÊTE: *Fournisseurs qui fournissent au moins un produit avec un coût supérieur au coût des produits fournis par Jean*

SQL:

```
SELECT DISTINCT P1.FNOM
FROM   FOURNITURE P1
WHERE  NOT EXISTS  (SELECT * FROM   FOURNITURE P2
                    WHERE  P2.FNOM = 'JEAN'
                    AND    P1.PRIX <= P2.PRIX)

SELECT DISTINCT FNOM FROM   FOURNITURE
WHERE  PRIX > ALL (SELECT PRIX FROM   FOURNITURE
                   WHERE FNOM = 'JEAN')
```

Division

Soit le schéma de relations

FOURNITURE(FNUM,PNUM,QUANTITE)
PRODUIT(PNUM,PNOM,PRIX)
FOURNISSEUR(FNUM,FNOM,STATUS,VILLE)

REQUÊTE: *Fournisseurs qui fournissent tous les produits*

ALGÈBRE:

$$\text{R1} := \pi_{FNUM,PNUM}(FOURNITURE) \div \pi_{PNUM}(PRODUIT)$$

$$\text{R2} := \pi_{FNOM}(FOURNISSEUR \bowtie R1)$$

SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  NOT EXISTS
       (SELECT *
        FROM  PRODUIT
        WHERE NOT EXISTS
              (SELECT *
               FROM  FOURNITURE
               WHERE FOURNITURE.FNUM = FOURNISSEUR.FNUM
               AND   FOURNITURE.PNUM = PRODUIT.PNUM))
```

Fonctions de Calcul

COUNT, SUM, AVG, MIN, MAX

REQUÊTE: *Nombre de Fournisseurs de Paris*

```
SELECT COUNT(*) FROM   FOURNISSEUR
WHERE  VILLE = 'Paris'
```

REQUÊTE: *Nombre de Fournisseurs qui fournissent actuellement des produits*

```
SELECT COUNT(DISTINCT FNOM) FROM   FOURNITURE
```

NOTE: La fonction COUNT(*) compte le nombre des n-uplets du résultat d'une requête sans élimination des dupliqués ni vérification des valeurs nulles. Dans le cas contraire on utilise la clause COUNT(UNIQUE . . .).

REQUÊTE: *Quantité totale de Briques commandées*

```
SELECT SUM (QUANTITE)
FROM   COMMANDES
WHERE  PNOM = 'Brique'
```

REQUÊTE: *Coût moyen de Briques fournies*

```
SELECT AVG (PRIX)
FROM   FOURNITURE
WHERE  PNOM = 'Brique'
```

ou

```
SELECT SUM (PRIX)/COUNT(PRI
  FROM FOURNITURE
 WHERE PNOM = 'Brique'
```

REQUÊTE: *Le prix des briques qui sont le plus chères.*

```
SELECT MAX (PRIX)
FROM   FOURNITURE
WHERE  PNOM = 'Briques';
```

REQUÊTE: *Fournisseurs des Briques au coût moyen des Briques*

```
SELECT FNOM
FROM   FOURNITURE
WHERE  PNOM = 'Brique' AND
       PRIX < (SELECT AVG(PRIX)
               FROM   FOURNITURE
               WHERE PNOM = 'Brique')
```

Opérations d'Agrégation

GROUP BY

REQUÊTE: *Nombre de fournisseurs par ville*

```
SELECT VILLE, COUNT(FNOM)
FROM   FOURNISSEUR
GROUP BY VILLE
```

LA BASE ET LE RESULTAT :

VILLE	FNOM
PARIS	TOTO
PARIS	DUPOND
LYON	DURAND
LYON	LUCIEN
LYON	REMI

VILLE	COUNT(FNOM)
PARIS	2
LYON	3

NOTE: La clause GROUP BY permet de préciser les attributs de partitionnement des relations declarées dans la clause FROM. Par exemple

on regroupe les fournisseurs par ville.

REQUÊTE: *Donner pour chaque produit fourni son coût moyen*

```
SELECT PNOM, AVG (PRIX)
FROM   FOURNITURE
GROUP BY PNOM
```

RÉSULTAT:

PNOM	AVG (PRIX)
BRIQUE	10.5
ARDOISE	9.8

NOTE: Les fonctions de calcul appliquées au résultat de régroupement sont directement indiquées dans la clause SELECT. Par exemple le calcul de la moyenne se fait par produit obtenu au résultat après le regroupement.

HAVING

REQUÊTE: *Produits fournis par deux ou plusieurs fournisseurs avec un coût supérieur de 100*

```
SELECT PNOM
FROM   FOURNITURE
WHERE  PRIX > 100
GROUP BY PNOM
HAVING COUNT(*) >= 2
```

AVANT LA CLAUSE HAVING

PNOM	FNOM	PRIX
BRIQUE	TOTO	105
ARDOISE	LUCIEN	110
ARDOISE	DURAND	120

APRÈS LA CLAUSE HAVING

PNOM	FNOM	PRIX
ARDOISE	LUCIEN	110
ARDOISE	DURAND	120

NOTE: La clause HAVING permet d'éliminer des partitionnements, comme la clause WHERE élimine des n-uplets du résultat d'une requête.

REQUÊTE: *Produits fournis et leur coût moyen pour les fournisseurs dont le siège est à Paris seulement si le coût minimum du produit est supérieur à 1000F*

```
SELECT PNOM, AVG(PRIX)
FROM   FOURNITURE, FOURNISSEUR
WHERE  VILLE = 'Paris' AND
       FOURNITURE.FNOM = FOURNISSEUR.FNOM
GROUP BY PNOM
HAVING MIN(PRIX) > 1000
```

ORDER BY

En général, le résultat d'une requête SQL n'est pas trié. Pour trier le résultat par rapport aux valeurs d'un ou de plusieurs attributs, on utilise la clause `ORDER BY`:

```
SELECT VILLE, FNOM, PNOM
  FROM FOURNITURE, FOURNISSEUR
 WHERE FOURNITURE.FNOM = FOURNISSEUR.FNOM
 ORDER BY VILLE, FNOM DESC
```

Le résultat est trié par les villes (ASC) et le noms des fournisseur dans l'ordre inverse (DESC).

Création et Mises à jour avec SQL

Valeurs NULL, par défaut (*)

- **Valeur NULL pour un attribut**: valeur spéciale qui représente une information inconnue (manquante, non fournie)
- attention ce n'est ni zéro, ni chaine vide
- pour forcer un attribut à toujours avoir une valeur, option *default*, exemple: default "manquant": si l'attribut n'a pas de valeur fournie, on la force à "manquant"

La valeur NULL (*)

1. $A\ \theta\ B$ est `inconnu` (ni vrai, ni faux) si la valeur de A ou/et B est NULL (θ est l'un de $=, <, \leq, >, \geq, \neq$).
2. $A\ op\ B$ est NULL si la valeur de A ou/et B est NULL (op est l'un de $+, -, *, /$).

Trois Valeurs de Vérité (*)

Trois valeurs de vérité: `vrai, faux` et **inconnu**

1. `vrai AND inconnu = inconnu`
2. `faux AND inconnu = faux`
3. `inconnu AND inconnu = inconnu`
4. `vrai OR inconnu = vrai`
5. `faux OR inconnu = inconnu`
6. `inconnu OR inconnu = inconnu`
7. `NOT inconnu = inconnu`

Soit le schéma de relation **FOURNISSEUR**(FNOM,STATUT,VILLE)

REQUÊTE: *Les Fournisseurs de Paris.*

SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  VILLE = 'Paris'
```

On ne trouve pas les fournisseurs avec VILLE = NULL !

Soit le schéma de relation **FOURNISSEUR**(FNOM,STATUT,VILLE)

REQUÊTE: *Fournisseurs dont l'adresse est inconnu.*

SQL:

```
SELECT FNOM
FROM   FOURNISSEUR
WHERE  VILLE IS NULL
```

NOTE: Le prédicat IS NULL (ou IS NOT NULL) n'est pas exprimable en algèbre relationnelle.

Exemple (*)

Soit le schéma de relation **EMPLOYE**(EMPNO,ENOM,DEPNO,SAL)

SQL:

```
SELECT E1.ENOM
  FROM EMPLOYE E1, EMPLOYE E2
 WHERE E1.SAL > 20000 OR
       E1.SAL <= 20000
```

Est-ce qu'on trouve les noms de tous les employés s'il y a des employés avec un salaire inconnu ?

Contraintes

Contraintes que l'on peut exprimer lors de la création de tables et que le système peut maintenir:

1. **NOT NULL**: un attribut doit toujours avoir une valeur
2. **PRIMARY KEY**: un (groupe d') attribut(s) constitue(nt) la clé primaire
3. **UNIQUE** un (groupe d') attribut(s) constitue(nt) une clé (secondaire).
4. **FOREIGN KEY (A) REFERENCES R** L'attribut A est la clé primaire de la table R (clé étrangère).
5. **CHECK**: contrainte sur la valeur

Création de Tables

On crée une table avec la commande **CREATE TABLE** :

```
CREATE TABLE Produit(pnom VARCHAR(20),
                     prix INTEGER,
                     PRIMARY KEY (pnom));

CREATE TABLE Fournisseur(fnom VARCHAR(20) PRIMARY KEY,
                         ville VARCHAR(16));

CREATE TABLE Fourniture (pnom VARCHAR(20) NOT NULL,
                         fnom VARCHAR(20) NOT NULL,
                    FOREIGN KEY (pnom) REFERENCES Produit,
                    FOREIGN KEY (fnom) REFERENCES Fournisseur
```

Création de Tables

```
CREATE TABLE ARTISTE (id INTEGER NOT NULL,
                      nom VARCHAR(30) NOT NULL,
                      anneeNaiss INTEGER, default '1900',
                        CHECK (anneeNaiss BETWEEN 1900 AND 2
                      film VARCHAR (30),
                      PRIMARY KEY (id),
                      UNIQUE(nom),
                      FOREIGN KEY (film) REFERENCES FILM)

CREATE TABLE FILM (idfilm VARCHAR (30) NOT NULL,...)
```

Intégrité référentielle

Le système vérifie les contraintes d'intégrité référentielle. Ses réactions dépendent des options choisies lors de la création des tables, par exemple:

1. si on insère un artiste avec l'attribut film renseigné, le film doit exister dans la table Film
2. si un film est supprimé dans la relation FILM, les nuplets qui réfèrent ce film dans la table ARTISTE ont l'attribut film mis à **NULL** (option *ON DELETE SET NULL*).
3. répercute une maj de l'id faite dans FILM, dans les nuplets qui réfèrent ce film dans ARTISTE (option *ON UPDATE CASCADE*).

Destruction de Tables

On détruit une table avec la commande **DROP TABLE** :

```
DROP TABLE Fourniture;
DROP TABLE Produit;
DROP TABLE Fournisseur;
```

Insertion de n-uplets

On insère dans une table avec la commande **INSERT** dont voici la syntaxe.

INSERT INTO $R(A_1, A_2, \ldots, A_n)$ **VALUES** $(v_1, v_2, \ldots v_n)$

Donc on donne deux listes : celles des attributs (les A_i) de la table et celle des valeurs respectives de chaque attribut (les v_i).

1. Bien entendu, chaque A_i doit être un attribut de R
2. Les attributs non-indiqués restent à **NULL** ou à leur valeur par défaut.
3. On doit toujours indiquer une valeur pour un attribut déclaré **NOT NULL**

Insertion : exemples

Insertion d'une ligne dans *Produit* :

INSERT INTO *Produit (pnom, prix)*
VALUES (*'Ojax', 15*)

Insertion de deux fournisseurs :

INSERT INTO *Fournisseur (fnom, ville)*
VALUES (*'BHV', 'Paris'*), (*'Casto', 'Paris'*)

Il est possible d'insérer plusieurs lignes en utilisant **SELECT**

INSERT INTO *NomsProd (pnom)*
SELECT DISTINCT *pnom* **FROM** *Produit*

Modification

On modifie une table avec la commande **UPDATE** dont voici la syntaxe.

UPDATE R **SET** $A_1 = v_1, A_2 = v_2, \ldots, A_n = v_n$
WHERE *condition*

Contrairement à **INSERT**, **UPDATE** s'applique à un ensemble de lignes.

1. On énumère les attributs que l'on veut modifier.
2. On indique à chaque fois la nouvelle valeur.
3. La clause **WHERE** *condition* permet de spécifier les lignes auxquelles s'applique la mise à jour. Elle est identique au **WHERE** du **SELECT**

Bien entendu, on ne peut pas violer les contraintes sur la table.

Modification : exemples

Mise à jour du prix d'Ojax :

UPDATE *Produit* **SET** *prix=17*
WHERE *pnom = 'Ojax'*

Augmenter les prix de tous les produits fournis par BHV par 20% :

UPDATE *Produit* **SET** *prix = prix*1.2*
WHERE *pnom* in (**SELECT** *pnom*
FROM *Fourniture*
WHERE *fnom = 'BHV'*)

Destruction

On détruit une ou plusieurs lignes dans une table avec la commande **DELETE**

DELETE FROM R
WHERE *condition*

C'est la plus simple des commandes de mise-à-jour puisque elle s'applique à des lignes et pas à des attributs. Comme précédemment, la clause **WHERE** *condition* est indentique au **WHERE** du **SELECT**

Destruction : exemples

Destruction des produits fournit par le BHV :

DELETE FROM *Produit*
WHERE *pnom* in (**SELECT** *pnom*
FROM *Fourniture*
WHERE *fnom = 'BHV*)

Destruction du BHV :

DELETE FROM *Fournisseur*
WHERE *fnom = 'BHV'*