Bases de Données Relationnelles

Cours SGBD - Module 6A - ENSTA

M. Scholl, B. Amann

http://cedric.cnam.fr/vertigo/Cours/ENSTA

Septembre 2001

INTRODUCTION

OBJECTIF DU COURS?

OBJECTIF: Comprendre et maitriser la technologie des systèmes de bases de données relationnelle:

- **Modélisation** des données
- **Interrogation** des données: évaluation des requêtes
- **Mises-à-jour** des données: concurrence et transactions
- **Intégration** des données: entrepôts de données, médiateurs
- **Architectures** et applications nouvelles: Web, portails, ...

Et tout **ça** en 21 heures?

PLAN

1. Introduction
2. Modèle Relationnel
3. Calcul et Algèbre Relationnel
4. SQL
5. Organisation Physique, Index
6. Algorithmes de Jointure
7. Optimisation des requêtes: l’exemple d’ORACLE.

BIBLIOGRAPHIE

Ouvrages en français

1. Date C.J, *Introduction aux Bases de Données*, Vuibert, 970 Pages, Janvier 2001
2. Carrez C., *Des Structures aux Bases de Données*, Masson

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*

Applications “SGBD”

1. **CLASSIQUES:**
   - Données de Gestion (salaires, stocks, réservations d’avions)
   - Applicationes Transactionnelles (gestion de comptes bancaires, centrales d’achat)
2. **MOINS CLASSIQUES:**
   - Documents électroniques
   - Données Spatiales
3. **A LA MODE:**
   - Données du Web: HTML, **XML**
   - Entrepôts de données
   - Portails d’entreprise

Comment Stocker et Manipuler les Données?

DONNÉES $\rightarrow$ BASE DE DONNÉES (B.D.)

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

LOGICIEL $\rightarrow$ SYSTÈME de GESTION de B.D. (S.G.B.D)

- 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. **utilisateurs** : administrateurs, programmeurs, non informaticiens, ...
2. **interfaces** : langages de programmation, saisie de données, génération de rapports, autres SGBD ...
3. **architectures** : données centralisées, distribuées, hétérogènes

En Résumé, un SGBD ...

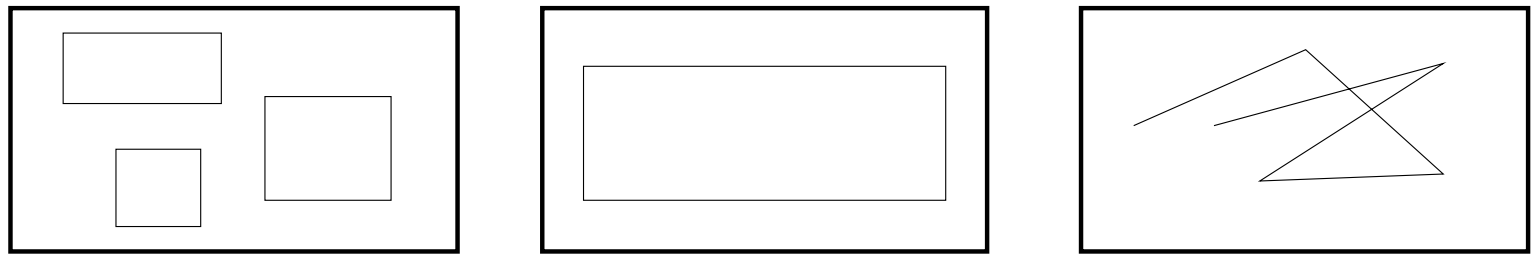
... est un **OUTIL GÉNÉRIQUE** qui doit répondre à des besoins très divers de gestion de **un GROS VOLUME D'INFORMATIONS**

- persistantes (années) et fiables (protection sur pannes)
- partageables (utilisateurs, programmes)
- manipulées indépendamment de leur organisation physique

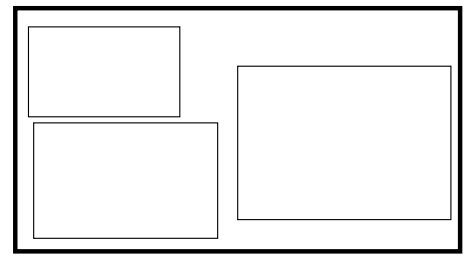
ARCHITECTURE d'un SGBD : ANSI-SPARC (1975)

NIVEAU EXTERNE

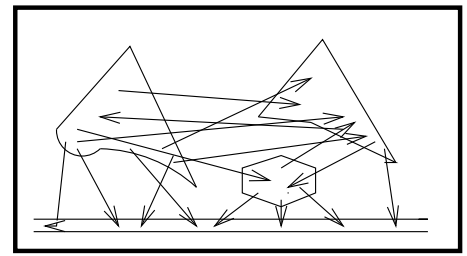
vue 1 **vue 2** **vue 3**



NIVEAU LOGIQUE



NIVEAU PHYSIQUE



FONCTIONNALITÉS d'un SGBD

Chaque niveau du SGBD réalise un certain nombre de fonctions :

NIVEAU PHYSIQUE

- Accès aux données sur mémoire secondaire (disques), index, ...
- partage de données et gestion de la concurrence d'accès
- reprise sur pannes (fiabilité)
- distribution et interopérabilité

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)

NIVEAU EXTERNE : Vues Utilisateurs

1. **Vue de la planification des salles :** pour chaque cours
   - Nom de Prof
   - Horaires et salles
2. **Vue de la paye :** un ensemble de Prof

   (nom, prénom, adresse, indice, nombre d'heures. . .)
3. **Vue du service de scolarité (suivi des élèves) :** . . .

Intégration de ces Vues

1. On laisse chaque usager avec sa vision du monde
2. PASSAGE DU NIVEAU EXTERNE AU NIVEAU LOGIQUE:
   On “intègre” l’ensemble de ces vues en une description **unique**: SCHÉMA LOGIQUE

Modélisation des données

Modèles de données

Un modèle de données est caractérisé par :

- une **structuration** des informations et
- des **opérations** sur ces structures

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-Relation

- 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 (index, stockage, ...) 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	RELATIONNEL ETENDU	nouvelles applications
	DATALOG (SGBD déductifs)	pas encore de marché
	ORIENTÉ-OBJET	navig. + déclaratif
Fin 90	HIÉRARCHIQUE (XML)	nouvelles applications Web

Opérations sur les données

Exemples d’opérations

Insérer les informations pour l’employé Jean

Augmenter le salaire de Jean de 10%

Supprimer les informations de Jean

Chercher les employés cadres

Chercher les employés du département comptabilité

Chercher le salaire moyen des employés comptables, avec deux enfants, nés avant 1960 et travaillant à Paris

Quels types d’opérations ?

4 types d’opérations classiques dans un SGBD:

1. La **création** (ou **insertion**) de données.
2. La **modification** de données.
3. La **destruction** de données.
4. La **recherche** de données.

Ces opérations correspondent à des manipulations de données (LMD) et sont appelés des **requêtes**. La plus complexe est la **recherche** en raison de la variété des critères.

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

Architecture Client-Serveur: plusieurs clients (utilisateurs, applications) 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.
- Donner une image cohérente des données si l'un fait des requêtes et l'autre des mises-à-jour.

Le but : éviter les blocages, tout en empêchant des modifications "anarchiques".

Concurrence d’accès: Exemple

Deux clients (A et B) partagent un compte bancaire (X): client A retire 100 euros du compte X et client B dépose 20 euros sur le même compte :

	compte X	client A	client B
1	500	U=read(X)	
2	500		V=read(X)
3	500	write(X,U-100)	
4	400		write(X,V+20)
5	520		

Qu’est-ce qui s’est passé?

Solutions: verrouillage, estampillage

Revenons aux *Utilisateurs* d'un SGBD

- **L'administrateur de la base**

  Rôle de l'administrateur/concepteur

  - discute avec les différents utilisateurs
  - conception d'un schéma logique (et des différentes vues)
  - conception du schéma physique
  - installation de la base et réglages fins (tuning)
  - gère l'évolution de la base (nouveaux besoins, utilisateurs)

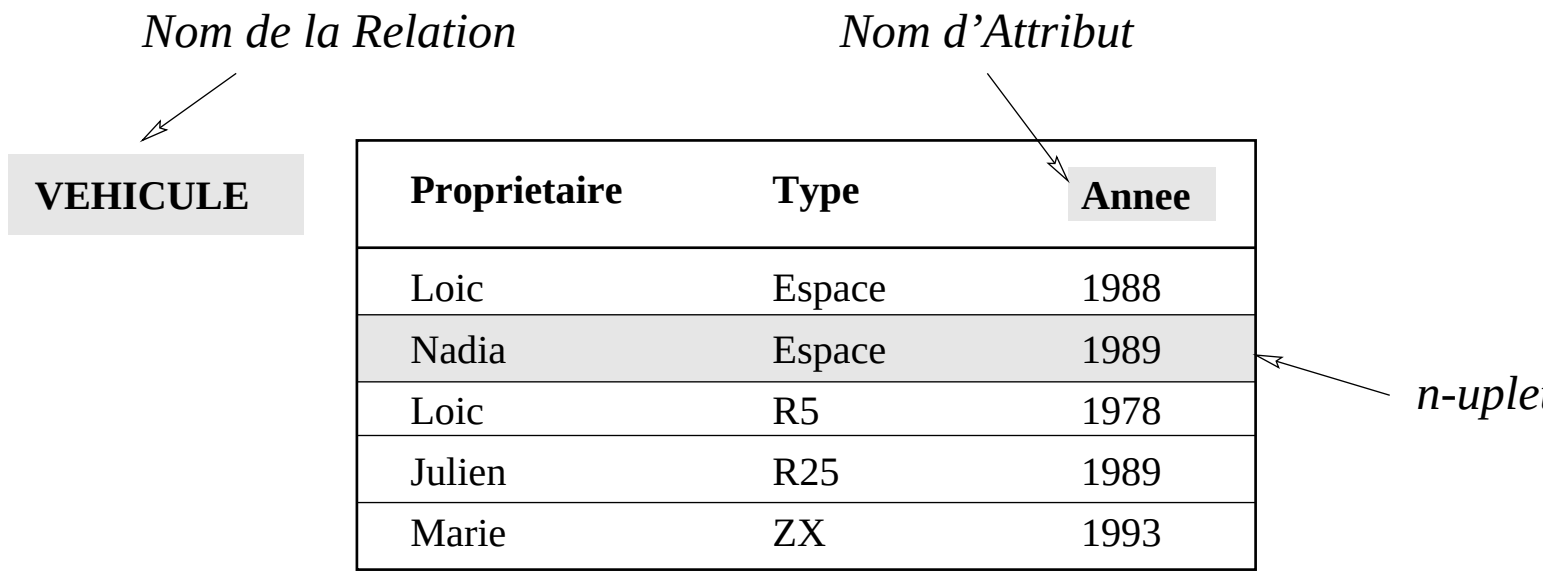
  Outils à sa disposition fournis par l'éditeur du SGBD

- **Utilisateur expert:** informaticien connaissant langages programmation et langages BD
- **Concepteur et programmeur d'application**
  écrit les applications pour des utilisateurs "naïfs"
- **Utilisateur naïf:** du non-spécialiste des SGBD au non-informaticien.

LE MODÈLE RELATIONNEL

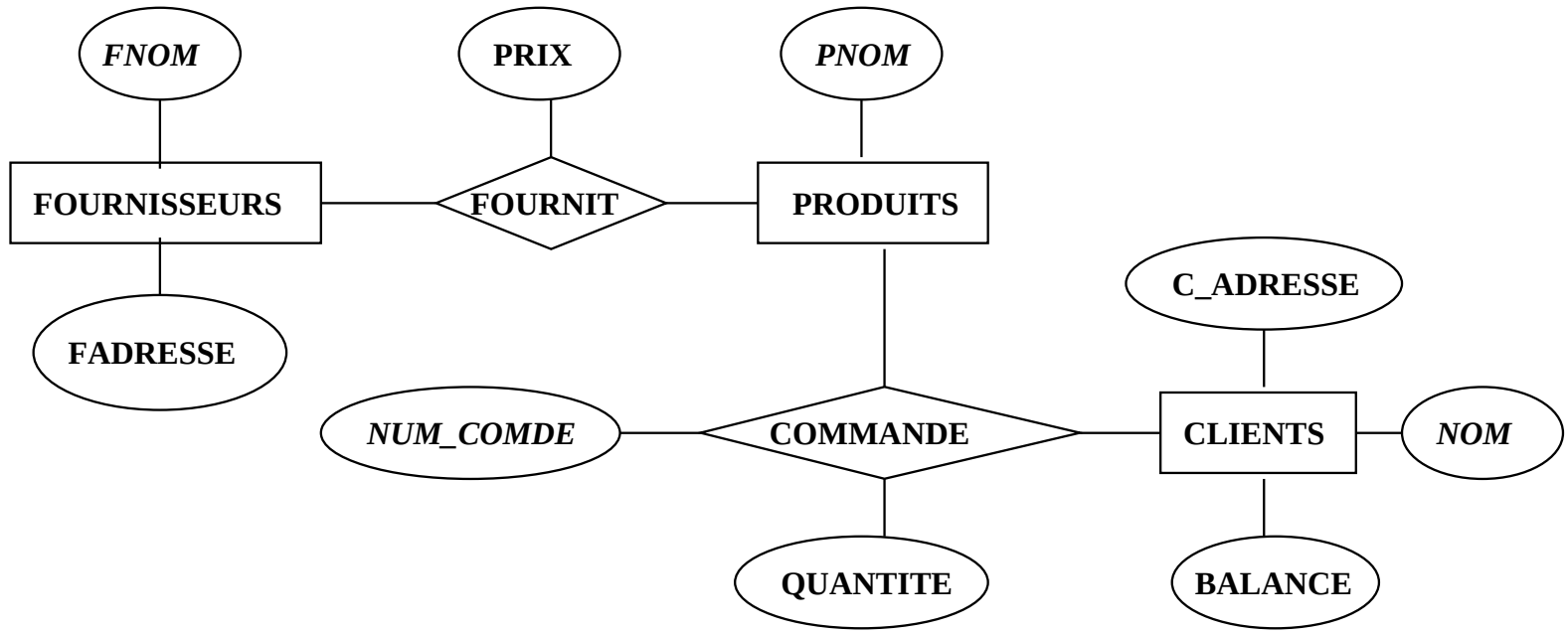
Présentation Générale

Exemple de Relation



Le Modèle Relationnel sur un Exemple

Schéma Entités-Relations (ER):



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
1	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

Terminologie et Définitions

Base de Données: Domaine, Nuplet, Relation

- Un DOMAINE est un ensemble (fini ou infini) de valeurs.
  Exemples : $Bool = \{0,1\}$, $Integer = \{0,1,-1,2,-2,\ldots\}$, $String$ =l'ensemble des chaînes de caractères, $String(10)$ =l'ensemble des chaînes de caractères de longueur 10.
- Un NUPLET est une liste de valeurs $(v_1,\ldots,v_n)$.
  Exemples: $('Abounayan','sable',10)$, $(1,'Jean','briques',5)$
- Le PRODUIT CARTÉSIEN $D_1 \times \ldots \times D_n$ entre des domaines $D_1,\ldots,D_n$ est l'ensemble de **tous** les n-uplets $(v_1,\ldots,v_n)$ où $v_i \in D_i$.
- Une RELATION est un sous-ensemble **fini** d'un produit cartésien $D_1 \times \ldots \times D_n$.
- Une BASE DE DONNÉES est un ensemble de relations.

Schéma de Bases de Données

- Un ATTRIBUT est défini par un nom et un domaine.
  Exemples: $NOM : String$, $QUANTITE : Integer$.
- Le SCHÉMA D'UNE RELATION R est défini par un nom et une liste d'attributs.
  Notation :
  $$R(A_1 : D_1, \ldots, A_n : D_n)$$
  ou plus simplement :
  $$R(A_1, \ldots, A_n)$$
- Un SCHÉMA DE BASES DE DONNÉES est un ensemble de schémas de relation.

Remarque: une relation ne peut pas avoir deux attributs du même nom et un attribut est toujours identifié à travers une relation. Ainsi, deux relations peuvent avoir des attributs de même noms avec des types différents (variables locales).

Instances d'un Schéma

- INSTANCE D'UN SCHÉMA DE RELATION : Une instance r d'un schéma de rélation $R(A_1 : D_1, \ldots, A_n : D_n)$ est un sous-ensemble fini du produit cartésien $D_1 \times \ldots \times D_n$: l'ensemble r est une relation.
- INSTANCE D'UN SCHÉMA DE BD: Une instance b d'un schéma de bases de données est un ensemble d'instances de ses schémas de relation: b est une base de données.

Remarque: une relation r est représentée sous forme d'une **table**. L'ordre des colonnes (attributs) ou des lignes n'a pas d'importance (ensemble).

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

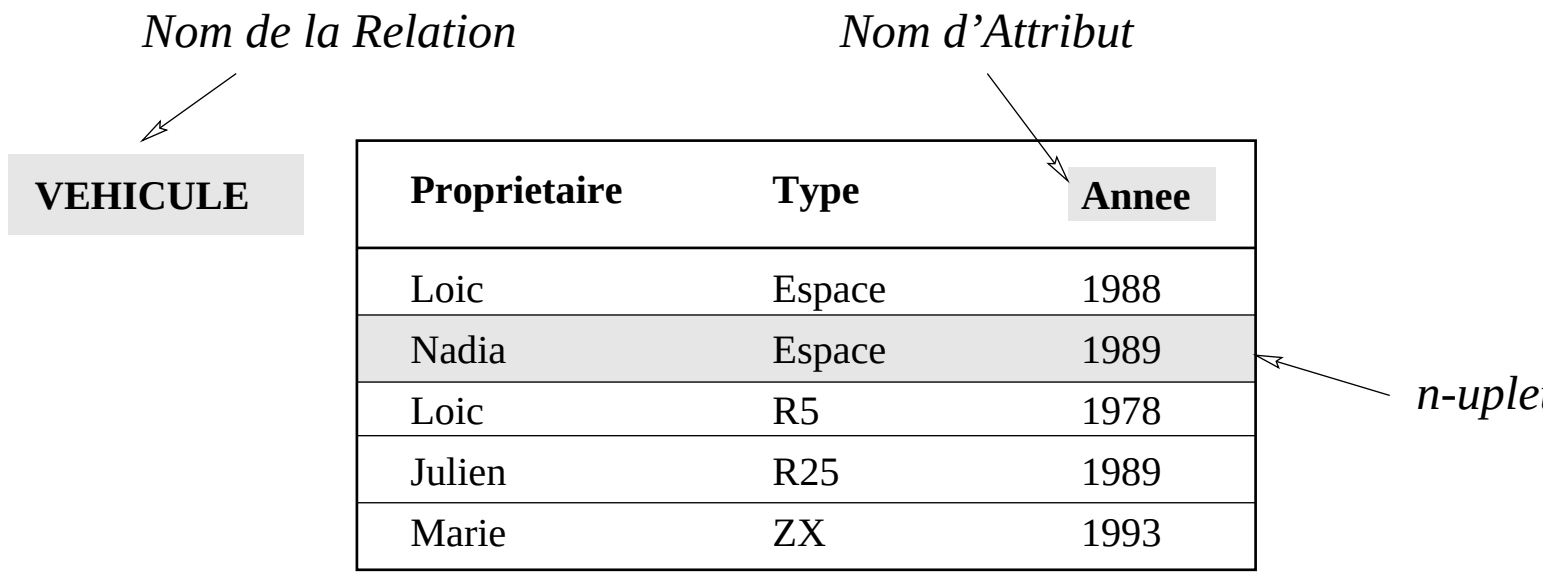
UNIVERS D'ATTRIBUTS :

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

RELATION UNIVERSELLE :

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

Revenons à l'exemple du début



- Traduction: Schéma conceptuel (ER) $\rightarrow$ Schéma logique (relationnel)
- Plus précisémment: Description du monde réel $\rightarrow$ Schéma conceptuel $\rightarrow$ Schéma logique

Objectif: Description du monde réel $=$ Schéma conceptuel $=$ Schéma logique?

Contraintes d'integrité

Objectif: Schéma logique + contraintes = Description du monde réel

- FOURNISSEUR(FNOM, FADRESSE)
- FOURNITURE(FNOM, PNOM, PRIX)
- COMMANDE(NUM_CMDE, NOM, PNOM, QUANTITÉ)

Exemples de contraintes:

1. Chaque fournisseur à un nom différent.
2. Chaque fournisseur à une seule adresse.
3. Chaque fournisseur fournit un produit à un seul prix.
4. Chaque commande est faite par un seul client.

- Contraintes 1 et 2: FNOM est une clé dans FOURNISSEUR
- Contraintes 1 et 3: (FNOM, PNOM) est une clé dans FOURNITURE
- Contrainte 4: NUM_CMDE “détermine” NOM dans COMMANDE :
  1. deux nuplets avec la même valeur pour NUM_CMDE ont la même valeur pour NOM.
  2. notation: NUM_CMDE $\rightarrow$ NOM (dépendance fonctionnelle)

  Attention: NUM_CMDE n’est pas une clé.

Remarque: les dépendances fonctionnelles ne réprésentent qu’un type de contraintes qui peuvent exister dans un schéma logique (dépendance d’inclusion, dépendances multi-valuées, ...).

Opérations et Langages

Opérations sur une Base de Données Relationnelle

- LANGAGE DE DÉFINITION DES DONNÉES (définition et MAJ du schéma) :
  - Création et destruction d'un schéma de relation (et de son instance).
  - Modification du schéma: ajout, suppression d'un attribut
  - Définition de contraintes.

- 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

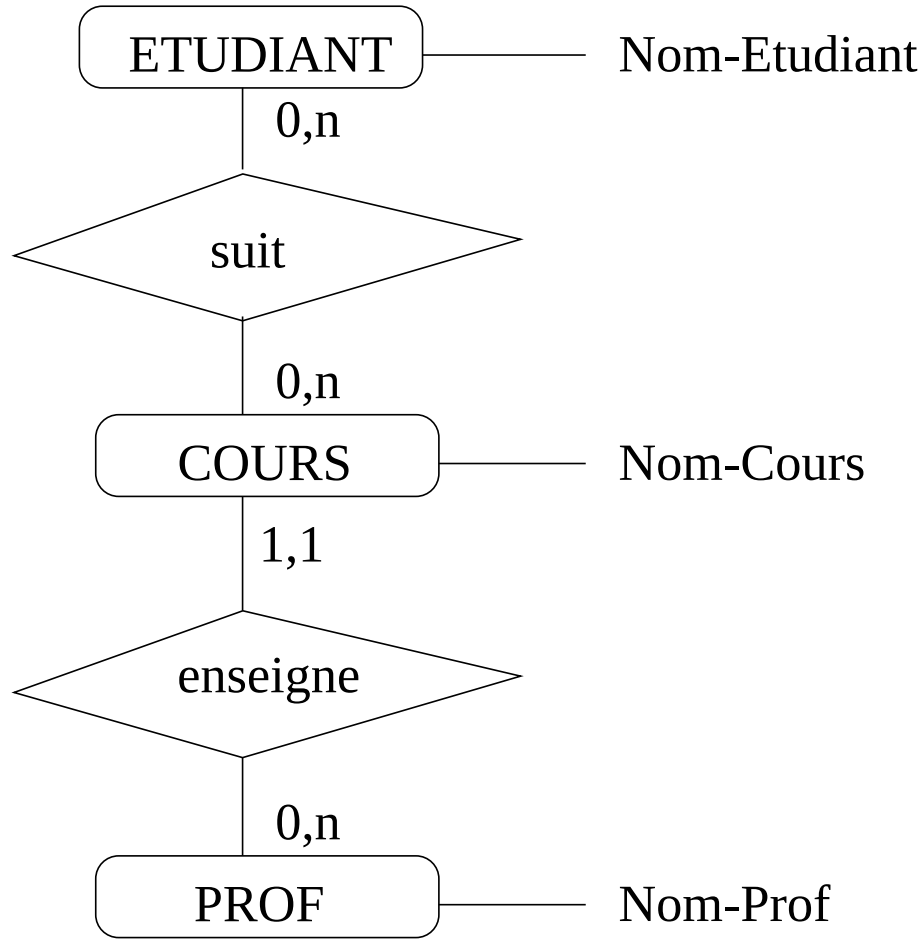
POUVOIR D'EXPRESSION : Qu'est-ce qu'on peut calculer ? Quelles opérations peut-on faire ?

Les langages de requête relationnels utilisent deux approches :

- calcul relationnel
- algèbre relationnelle

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

LES DÉPENDANCES FONCTIONNELLES


ETUDIANT
Nom-Etudiant
0,n
suit
0,n
COURS
Nom-Cours
1,1
enseigne
0,n
PROF
Nom-Prof

Exemples : Relation PERSONNE

PERSONNE	NOSS	NOM	VILLE
	123	toto	Paris
	324	mimi	Paris
	574	toto	Marseilles

Qu'est-ce qu'on peut dire sur la table PERSONNE?

- "Connaissant le numéro de sécurité sociale `NOSS`, je connais le `NOM`"
- L'attribut `NOSS` *détermine* l'attribut `NOM`
- L'attribut `NOM` *dépend fonctionnellement* de l'attribut `NOSS`
- Attention : Connaissant le `NOM` on ne connaît pas le `NOSS` : $\texttt{NOM} \not\rightarrow \texttt{NOSS}$

NOTATION : $\texttt{NOSS} \rightarrow \texttt{NOM}$

Relation COURS

COURS	NOM-COURS	PROF	ETUDIANT
	BDPI	MATHELOT	123
	BDPI	MATHELOT	324
	BDPI	MATHELOT	574
	IPA	HARDIN	123

- "Un Cours a un seul Professeur"
- `NOM-COURS` $\rightarrow$ `PROF`
- 2 nuplets qui ont même valeur pour `NOM-COURS` ont même valeur pour `PROF`

Clés, Superclés et Clés étrangères

Relation: PERSONNE (NOSS, NOM, PRENOM, ADRESSE)

NOSS $\rightarrow$ NOM, NOSS $\rightarrow$ PRENOM, NOSS $\rightarrow$ ADRESSE

- Le NOSS est une *clé*.

NOSS NOM $\rightarrow$ PRENOM

NOSS NOM est une *superclé* (NOM est redondant)

Relation: COURS (NOM-COURS, PROF, ETUDIANT)

Si v est une valeur de l'attribut ETUDIANT dans la table Cours, alors v est une clé dans la table personne.

- ETUDIANT est une clé étrangère (dépendance d'inclusion) entre la table COURS et la table PERSONNE.

Définitions

a) Dépendance fonctionnelle

Soit R(U) un schéma de relation, r une relation de schéma R, $X \subset U$, $Y \subset U$ deux sous-ensembles d'attributs de R. La *dépendance fonctionnelle*

$$X \rightarrow Y$$

est vrai dans r, ssi (si et seulement si) tous les nuplets de r qui ont même valeur pour (tous) les attributs de X, ont même valeur pour (tous) les attributs de Y.

Exemple :

```
ADRESSE(RUE, NUMERO, VILLE, CODE-P)
X = CODE-P
Y = VILLE
```

b) Superclé

Soit R(U) un schéma et X $\subset$ U un sous ensemble d'attributs.

X est une *superclé* de r de schéma R, si X $\rightarrow$ U.

Exemple :

NOSS NOM $\rightarrow$ NOSS NOM PRENOM ADRESSE

NOSS NOM est une superclé

c) Clé

X est une clé, si :

1. X est une superclé : X $\rightarrow$ U
2. il n'existe pas Y $\subset$ X, tel que Y $\rightarrow$ U

Exemple :

RUE NUMERO VILLE $\rightarrow$ RUE NUMERO VILLE CODE-P

RUE NUMERO VILLE est une clé (pourquoi?)

Calcul d'une Clé: Exemple

COURS(Nomc, Heure, Salle, Prof)

$\mathcal{F} = \mathtt{N} \rightarrow \mathtt{HS},\ \mathtt{HS} \rightarrow \mathtt{P}$

Nous pouvons prouver à partir de $\mathcal{F}$ que si on connaît le nom de cours, on connaît aussi le nom du professeur (`Nomc` est une clé) :

1. $\mathtt{N} \rightarrow \mathtt{HS}$: deux nuplets qui partagent le nom de cours partagent également l'horaire et la salle
2. $\mathtt{HS} \rightarrow \mathtt{P}$: deux nuplets qui partagent l'horaire et la salle partagent également le nom du professeur
3. DF 1 et 2 impliquent, que deux nuplets qui partagent le nom de cours partagent également le nom du prof : $\mathtt{N} \rightarrow \mathtt{P}$

On peut définir des propriétés (règles) sur les DF qui permettent de déduire d'autres DF.

Propriétés des Dépendances Fonctionnelles

(Axiomes d'Armstrong)

1) Réflexivité

si $X \subseteq Y$ alors $Y \rightarrow X$ (pour tous les ensembles d'attributs X et Y)

Exemple :

$\texttt{NOM VILLE} \rightarrow \texttt{NOM}$

Trivial : "Deux personnes qui ont le même nom et habitent la même ville, ont le même nom."

2) Transitivité

si X $\rightarrow$ Y et Y $\rightarrow$ Z, alors X $\rightarrow$ Z

Exemple :

R(NOSS, CODE-P, VILLE)

{NOSS $\rightarrow$ CODE-P, CODE-P $\rightarrow$ VILLE} $\Rightarrow$ NOSS $\rightarrow$ VILLE

"Si on connaît le code postale à partir du numéro de sécurité sociale et la ville à partir du code postale, on connaît la ville à partir du numéro de sécurité sociale."

3) Augmentation

X $\rightarrow$ Y $\Rightarrow$ XZ $\rightarrow$ YZ

Exemple :

NOSS $\rightarrow$ CODE-P $\Rightarrow$ NOSS VILLE $\rightarrow$ CODE-P VILLE

4) Union et décomposition

$$\{ X \rightarrow A, X \rightarrow B \} \Leftrightarrow X \rightarrow AB$$

5) Pseudotransitivité

$$\{ X \rightarrow Y, WY \rightarrow Z \} \Rightarrow WX \rightarrow Z$$

REMARQUE : Union, décomposition et pseudotransitivité peuvent être déduites des autres axiomes

Clôture d'un ensemble de dépendances fonctionnelles

De l'ensemble des dépendances fonctionnelles données par l'analyse du monde réel, en utilisant les propriétés ci-dessus, appelées *Axiomes d'Armstrong*, on peut en déduire d'autres :

Exemples :

1) R(A,B,C,D)

$\mathcal{F}$ = { A → B, B → C }

Par transitivité, on déduit A → C

Notation : $\mathcal{F} \models$ A → C

2) R(Cours, Prof, Heure, Salle, Eleve, Note)

$\mathcal{F}$ = { C $\rightarrow$ P, HS $\rightarrow$ C, HP $\rightarrow$ S, CE $\rightarrow$ N, HE $\rightarrow$ S }

$\mathcal{F} \models$ HS $\rightarrow$ P, HE $\rightarrow$ C,P,N

Donc HE est une **clé**. Pourquoi?

L'union de $\mathcal{F}$ et de toutes les dépendances ainsi déduites est appelée **clôture**, ou **couverture** de $\mathcal{F}$ et notée $\mathcal{F}^+$.

Nouvelle définition d'une Superclé

Soit $\mathsf{R(U)}$ un schéma de relation, $\mathcal{F}$ un ensemble de dépendances fonctionnelles et $\mathsf{X} \subseteq \mathsf{U}$ un ensemble d'attributs. X est une superclé de R si pour tout $\mathsf{A} \in \mathsf{U}$

$$\mathcal{F} \models \mathsf{X} \rightarrow \mathsf{A}$$

ou encore si:

$$\mathsf{X} \rightarrow \mathsf{A} \in \mathcal{F}^{+}$$

Calculer $\mathcal{F}^{+}$ à partir de $\mathcal{F}$ peut être très long.

Montrer que $\mathcal{F} \models \mathsf{X} \rightarrow \mathsf{A}$ est plus facile.

Calcul de la clé d'une relation

Soit $\mathsf{R(U)}$ un schéma de relation et $\mathsf{X} \subseteq \mathsf{U}$ un ensemble d'attributs.

- On définit X^+ comme ensemble des attributs A tels que $\mathcal{F} \models \mathsf{X} \rightarrow \mathsf{A}$
- Si A appartient à X^+, alors par définition $\mathcal{F} \models \mathsf{X} \rightarrow \mathsf{A}$
- X^+ est l'ensemble des attributs fonctionnellement dépendants de X.

Pour calculer une clé on utilise l'algorithme suivant :

1. On cherche un X tel que $X^+ = \mathsf{U} \Rightarrow \mathsf{X}$ est une superclé
2. X est une clé, s'il n'existe pas $\mathsf{Y} \subset \mathsf{X}$ tel que $Y^+ = \mathsf{U}$

Calcul de X^+

Étape 1: On part de X

Pour toute $\mathsf{Y} \rightarrow \mathsf{A}$, telle que $\mathsf{Y} \subseteq \mathsf{X}$, $\mathsf{Y} \rightarrow \mathsf{A} \in \mathcal{F}$, on rajoute A à X.

On obtient X^1.

Étape i: On part de X^{i-1}

Pour toute $\mathsf{Y} \rightarrow \mathsf{A}$, telle que $\mathsf{Y} \subseteq X^{i-1}$, $\mathsf{Y} \rightarrow \mathsf{A} \in \mathcal{F}$, on rajoute A à X^{i-1}.

On obtient X^i.

On s'arrête quand on ne trouve plus de nouvelle DF (point fixe) :

$$X^+ = X^{i+1} = X^i$$

Exemples

1. Montrer que `HE` est une clé pour `R(CHENSP)` avec l'ensemble $\mathcal{F}$ de DF donné ci-dessus
2. Pour la relation `ADRESSE(VILLE, RUE, NUMERO, CODE_P)` ci-dessus, montrer que `VILLE RUE NUMERO` est une clé. Quelle est l'autre?

Contraintes d'intégrité

La liste des attributs est insuffisante pour décrire la sémantique du monde réel.

Il existe plusieurs types de contraintes sur les nuplets :

1. dépendances (fonctionnelles, d'inclusion, multivaluées, etc.)
2. contraintes qui dépendent du domaine d'un attribut : Taille $<$ 2m10, année $<$ 2000
3. etc.

Ce sont les dépendances qui permettent la *conception d'un bon schéma*.

ANOMALIES DE MISE À JOUR

Exemple

Soit le schéma S1 :

```
FOURNISSEUR(FNOM, FADRESSE)
FOURNITURE(FNOM, PNOM, PRIX)
```

et l'ensemble de DF : $\mathcal{F} = \{$`FNOM` $\rightarrow$ `FADRESSE`, `(FNOM PNOM)` $\rightarrow$ `PRIX`$\}$

Supposons qu'on remplace S1 par le schéma S2 :

```
R(FNOM, FADRESSE, PNOM, PRIX)
```

Anomalies

R (FNOM, FADRESSE, PNOM, PRIX)

$\mathcal{F} = \{$FNOM $\rightarrow$ FADRESSE, (NOM PNOM) $\rightarrow$ PRIX$\}$

Quelle est la clé de R?

TOTO	LYON	BRIQUES	1000
DUPONT	ROUEN	BRIQUES	900
TOTO	LYON	BETON	400

1) REDONDANCE : l'adresse d'une personne apparaît plusieurs fois.

2) MAJ : si on modifie l'adresse dans un nuplet, il faut le faire dans les autres.

3) SUPPRESSION : si `DUPONT` ne fournit plus de `BRIQUES`, on supprime le 2e nuplet, on perd toute info sur `DUPONT`

4) INSERTION : on ne peut insérer un nouveau fournisseur et son adresse, si on ne connaît pas au moins un produit qu'il fournit

TOTO	PARIS	BRIQUES	1000
TOTO	LYON	BETON	400
DURAND	NICE		

$\Longrightarrow$ LE SCHÉMA INITIAL S1 EST "MEILLEUR"

Qualités d'un bon schéma

1. Éviter les anomalies $\Longrightarrow$ décomposition
2. La décomposition doit conserver la même information
   La jointure d'une relation
   `f1` *de schéma* `FOURNISSEUR(FNOM, FADRESSE)`
   et d'une relation
   `f2` *de schéma* `FOURNITURE(FNOM, PNOM, PRIX)`
   obtenues par décomposition d'une relation
   `r` *de schéma* `R(FNOM, FADRESSE, PNOM, PRIX)`
   doit redonner `r`.
3. La décomposition doit conserver les mêmes contraintes (DF). La décomposition de `R` en `R1(FNOM, FADRESSE, PRIX)` et `R2(PNOM, PRIX)` ne préserve pas les DF. Pourquoi?

FORMES NORMALES ET DÉCOMPOSITION

Relation en première forme normale (1FN)

- Tous les attributs sont atomiques (*élémentaires*)
- Relations telle qu'on les connaît.
- **Relation non normalisée, non en 1e FN** :
  Certains attributs sont des ensembles de valeurs, des relations elles même

Relation 1FN

NOTES (COURS	ETUDIANT	NOTE)
BDB	Toto	15
BDB	Lulu	17
BDB	Lili	0
ARCHI	Lili	20
ARCHI	Toto	0

Relation N1FN

NOTES (COURS	PERF (ETUDIANT	NOTE))
BDB	Toto	15
	Lulu	17
	Lili	0
ARCHI	Lili	20
	Toto	0

Relation en 3e forme normale

- **2e forme normale** :

  Purement historique
- **3e forme normale : 3FN**
  - évite la plupart des anomalies

  le but du jeu : décomposer une relation (1FN) en un ensemble de relations 3FN

3FN : Définition

Définition : Une relation R est en 3FN, si quelle que soit la DF $X \rightarrow A$ de $\mathcal{F}^+$ où

- A est un seul attribut et
- A n'est pas l'un des attributs de X,

- soit X est une *super*clé (contient une clé),
- soit A appartient à l'une des clés.

En fait, il n'est pas nécessaire de vérifier *toutes* les DF de $\mathcal{F}^+$.

Il suffit de vérifier celles de $\mathcal{F}$!

Exemples: 3e Forme Normale

1) Poste(Ville, Rue, Code)

$\mathcal{F} = \{ \texttt{VR} \rightarrow \texttt{C}, \ \texttt{C} \rightarrow \texttt{V} \}$

Clés : {V, R} et {R, C}

R est en 3FN.

2) FOURNITURE(NOMF, ADR, NOMP, PRIX)

$\mathcal{F} = \{ \texttt{NOMF} \rightarrow \texttt{ADR}, \ \texttt{NOMF, NOMP} \rightarrow \texttt{PRIX} \}$

Clé : {NOMF, NOMP}

FOURNITURE n'est pas en 3FN.

3) PLANNING(Cours, Heure, Salle)

$\mathcal{F}$ = { SH → C, C → S }

Clés : {S,H} et {C,H}

PLANNING est en 3FN.

4) R(A, B, C, D)

$\mathcal{F}$ = { AB → C, B → D, BC → A }

Clés : {A, B}, {B, C}

R n’est pas en 3FN.

Décomposition sans perte d'information (SPI)

Exemple

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

et $\mathcal{F} = \{ \mathtt{A} \rightarrow \mathtt{B} \}$

Décomposons en :

R1	A	B
	a	b
	c	b

R2	B	C
	b	c
	b	a
	b	d

R1 = $\pi_{A,B}$(R)

R2 = $\pi_{B,C}$(R)

R′ = R1 ⋈ R2 ≠ R :

R′	A	B	C
	a	b	c
	a	b	a
	a	b	d
	c	b	c
	c	b	a
	c	b	d

La décomposition de R en R1 et R2 est avec perte d'informations.

La jointure crée des nuplets qui n'existaient pas dans R.

Décomposons R′ maintenant en :

R1′ (A	B)
a	b
c	b

R2′ (A	C)
a	c
a	a
c	d

R″ = R1′ ⋈ R2′ = R′ :

R′′ (A	B	C)
a	b	c
a	b	a
c	b	d

Cette décomposition est *sans perte d'information* (SPI)

Il faut qu'après la jointure, on retrouve la même information qu'avant la décomposition.

Définition

Une décomposition de R en R1, R2, ..., Rk par rapport à un ensemble de DF $\mathcal{F}$ est SPI (sans perte d'information), ssi quelle que soit r de schéma R *satisfaisant* $\mathcal{F}$, on a :

$$r = \pi_{R1}(r) \bowtie \pi_{R2}(r) \ldots \bowtie \pi_{Rk}(r)$$

Théorème :

Si (R1, R2) est une décomposition de R et $\mathcal{F}$ un ensemble de DF, alors (R1, R2) est SPI par rapport à $\mathcal{F}$, ssi :

$$R1 \cap R2 \rightarrow R1 - R2$$

ou

$$R1 \cap R2 \rightarrow R2 - R1$$

est une dépendance de $\mathcal{F}^+$.

Exemples

$R\ (A,\ B,\ C)$

$\mathcal{F} = \{\ A \rightarrow B\ \}$

1) R1(A, B), R2(B, C)

$AB \cap BC = B$

$AB - BC = A$

$BC - AB = C$

Il n'y a ni la DF $B \rightarrow A$, ni la DF $B \rightarrow C$ dans $\mathcal{F}^+$

$\Rightarrow$ **Décomposition avec perte d'information**

2) R1(A, B), R3(A, C)

AB $\cap$ AC = A

AB - AC = B

A $\rightarrow$ B est dans $\mathcal{F}$ ($\mathcal{F}^+$).

$\Rightarrow$ **Décomposition sans perte d'information**

Décomposition qui préserve les dépendances fonctionnelles

Définitions

1. Projection d'un ensemble de dépendances sur $\mathtt{Z} \subset \mathtt{U}$

   $\pi_Z(\mathcal{F}) = \{X \rightarrow Y \in \mathcal{F}^+ \mid XY \subseteq Z\}$

   Exemple : $\mathtt{R(A,B,C,D)}$, $\mathcal{F}$ = { $\mathtt{AB} \rightarrow \mathtt{C}$, $\mathtt{C} \rightarrow \mathtt{A}$, $\mathtt{A} \rightarrow \mathtt{D}$ }

   $\pi_{ABC}(\mathcal{F})$ = { $\mathtt{AB} \rightarrow \mathtt{C}$, $\mathtt{C} \rightarrow \mathtt{A}$ }

2. Décomposition qui préserve les DF de $\mathcal{F}$

   Soit Δ = $\mathtt{(R1,...,Rk)}$ une décomposition, et $\mathcal{F}$ un ensemble de DF.

   Δ préserve les DF de $\mathcal{F}$, si on peut retrouver toutes les DF de $\mathcal{F}^+$ à partir de l'union $\mathcal{G}$ de toutes les DF projetées de $\mathcal{F}$ dans $\pi_{R1}(\mathcal{F}), \ldots, \pi_{Rk}(\mathcal{F})$:

   $\mathcal{G}^+ = \mathcal{F}^+$

Exemples

R(A,B,C,D)

$\mathcal{F} = \{ \mathtt{AB} \rightarrow \mathtt{C}, \mathtt{C} \rightarrow \mathtt{A}, \mathtt{A} \rightarrow \mathtt{D} \}$

Δ = (ABC, BD) ne préserve pas les DF de $\mathcal{F}$

Δ = (ABC, AD) préserve les DF de $\mathcal{F}$

R(A,B,C)

$\mathcal{F} = \{ \mathtt{A} \rightarrow \mathtt{B}, \mathtt{B} \rightarrow \mathtt{A}, \mathtt{A} \rightarrow \mathtt{C} \}$

Δ = (AB, BC) préserve les DF de $\mathcal{F}$

R(A, B, C, D)

$\mathcal{F}$ = { A → B, B → C, AB → D }

La décomposition

R1(AC)

R2(AB)

R3(CD)

ne préserve pas les DF de $\mathcal{F}$. Pourquoi?

Décomposition d'une relation en relations 3FN

Étant donné un schéma `(R, F)` non en 3FN, i.e. avec des anomalies, on veut une décomposition de `R` :

1. en relations 3FN
2. qui soit SPI
3. qui préserve les DF de $\mathcal{F}$

Remarque :

- une décomposition SPI ne préserve pas forcément les DF et inversement
- le résultat ne donne pas forcément des relations 3FN

Théorème : Toute relation en 1FN possède une décomposition en relations 3FN qui soit SPI et préserve les dépendances fonctionnelles.

Algorithme de décomposition

On suppose que $\mathcal{F}$ est une couverture minimale

1. Pour chaque $\mathtt{X} \rightarrow \mathtt{A} \in \mathcal{F}$, créer une relation de schéma `(XA)`.
2. Si aucune des clés n'est contenue dans l'un des schémas créés dans l'étape 1, rajouter une relation de schéma `(Y)`, où `Y` est une clé.
3. Si après l'étape 1, il existe une relation `R1` dont le schéma `(X1A1)` est contenu dans le schéma `(X2A2)` d'une autre relation `R2`, supprimer la relation `R1`.
4. Remplacer les relations `(XA1),...,  (XAk)` (correspondant à des dépendances ayant même membre gauche) par une relation unique : `(XA1...Ak)`.

Exemples

1) R(A,B,C,D)

$\mathcal{F}$ = { AB → C, B → D, C → A }

Clés : AB, BC

- Étape 1 : R1(ABC) R2(BD) R3(CA)
- Étape 2 : Pas la peine de rajouter une relation de schéma la clé AB :

  AB est contenue dans R1
- Étape 3 : Supprimer R3 : CA ⊂ ABC

Bonne décomposition : R1(ABC) R2(BD)

On peut vérifier que R1 et R2 sont en 3eFN.

2) R(A,B,C,D,E)

$\mathcal{F} = \{ \mathtt{AB} \rightarrow \mathtt{C}, \mathtt{C} \rightarrow \mathtt{D}, \mathtt{C} \rightarrow \mathtt{A} \}$ **Clés** : ABE, BCE

- Étape 1 : R1(ABC) R2(CD) R3(CA)
- Étape 2 : On rajoute une relation de schéma pour la clé ABE : R4(ABE)
- Étape 3 : Supprimer R3 : CA $\subset$ ABC

Bonne décomposition : R1(ABC) R2(CD) R4(ABE)

R3 n'a pas de dépendance. Quelles sont les dépendances des autres?

Autre solution :

- Étape 4 : On remplace R2 et R3 de l'étape 1 par une relation de schéma (CAD)

Autre bonne décomposition : R1(ABC) R2′(CAD) R4(ABE)

Que se passe-t-il si on avait choisi la clé CBE?

3) R(A,B,C,D)

$\mathcal{F} = \{$ AB $\rightarrow$ C, C $\rightarrow$ D, C $\rightarrow$ A, AB $\rightarrow$ D $\}$

Clés : BA, BC

La relation n'est pas en 3e Forme Normale. Pourquoi?

- Étape 1 : R1(ABC) R2(CD) R3(CA) R4(ABD)
- Étape 2 : on ne rajoute pas de relation : clé AB $\subseteq$ R1(ABC)
- Étape 3 : Supprimer R3 : CA $\subseteq$ ABC
- Étape 4 : On remplace R1 et R4 par R5(ABCD) $\Rightarrow$ on peut supprimer R2.

Décomposition : R5(ABCD)

Cette décomposition n'est pas en 3e Forme Normale. Où est le problème?

Forme Normale Boyce-Codd (BCNF)

Des anomalies subsistent en 3FN.

Exemple : `Poste(Ville,Rue,Code)`, $\mathcal{F}$ = { `VR` $\rightarrow$ `C`, `C` $\rightarrow$ `V` }

Clés : `VR, RC`

Poste	(Ville	Rue	Code)
	Paris	St Michel	75005
	Paris	Champollion	75005

$\Rightarrow$ Redondance entre le code et la ville.

Définition : Une relation est en forme normale de *Boyce-Codd* (BCNF), si quelle que soit la dépendance de $\mathcal{F}$, le membre de gauche est une clé.

Intérêt : On a éliminé toutes les anomalies

Remarque : Toute relation BCNF est en 3FN

Malheureusement, il n'existe pas toujours une décomposition en relations BCNF

- qui soit SPI
- qui préserve les DF

L'exemple de la poste

`Poste(Ville,Rue,Code)`, $\mathcal{F}$ = { `VR` $\rightarrow$ `C`, `C` $\rightarrow$ `V` }

Clés : `VR, RC`

`R` est 3FN mais n'est pas BCNF (dans `C` $\rightarrow$ `V`, `C` n'est pas une clé)

Poste	Ville	Rue	Code
	Sevres	de Gaulle	92310
	Chaville	de Gaulle	92370

La décomposition `R1(Ville,Code), R2(Rue,Code)` évite la redondance `Ville,Code`, elle est SPI, mais elle ne préserve pas la dépendance `VR` $\rightarrow$ `C`

```
R1 ( Ville     Code  )     R2 ( Rue        Code  )
     Sevres    92310              de Gaulle 92310
     Chaville  92370              de Gaulle 92370
```

L'insertion `Sevres de Gaulle 92190`, c.a.d. `Sevres 92190` et `de Gaulle 92190` respecte `C` $\rightarrow$ `V` mais ne respecte plus `VR` $\rightarrow$ `C`

```
R1 ( Ville     Code  )     R2 ( Rue        Code  )
     Sevres    92310              de Gaulle  92310
     Chaville  92370              de Gaulle  92370
     Sevres    92190              de Gaulle  92190
```

ALGÈBRE RELATIONNELLE

Algèbre Relationnelle

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

Opérations de l'Algèbre Relationnelle

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
→	a	b	c
	d	a	b
	c	b	d
→	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

$\Longrightarrow$ *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

$|R|$

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

S

$|S|$

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

$\Rightarrow$

$\mathbf{R \times S}$

$| R | \times | S |$

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

$\Rightarrow$

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 $\times$ **S**

	R.A	R.B	S.A	S.B	D
	1	a	1	a	b
$\rightarrow$	**1**	**a**	**2**	**c**	b
$\rightarrow$	**1**	**a**	**4**	**a**	a
$\rightarrow$	**1**	**b**	**1**	**a**	b
$\rightarrow$	**1**	**b**	**2**	**c**	b
$\rightarrow$	**1**	**b**	**4**	**a**	a
$\rightarrow$	**4**	**a**	**1**	**a**	b
$\rightarrow$	**4**	**a**	**2**	**c**	b
	4	a	4	a	a

$\Downarrow$

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 $\times$ **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

$\Rightarrow$

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

	A	B
$\rightarrow$	a	b
	a	c
$\rightarrow$	d	e
	a	e
	f	g

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 : $R \rhd\!\!< S$
- SCHÉMA DE $T = R \rhd\!\!< S$: $T(A_1,\ldots,A_m,X_1,\ldots,X_k)$
- VALEUR DE $T = R \rhd\!\!< S$: 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 $\ltimes$ **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: Exemple

R

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

S

C	D
1	b
2	b
4	a

$\Rightarrow$

R ÷ S

A	B	E
1	a	a
3	c	b

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

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

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

Et si les attributs sont connus par leur rang?

On peut dans les opérations de l'algèbre indiquer un attribut par son rang au lieu de son nom.

Dans ce cas l'ordre des attributs dans une relation a de l'importance.

L'exemple de la sélection :

$\sigma_{\&1='PARPAING'}COMMANDES$ au lieu de
$\sigma_{PNOM='PARPAING'}COMMANDES$

Le premier attribut &1 correspond a l'attribut $PNOM$

CALCUL RELATIONNEL

Exemple : la base de données CINÉMA

Films	Titre	Metteur en Scene	Acteur
	Jules et Jim	F. Truffaut	J. Moreau
	Jules et Jim	F. Truffaut	O. Werner
	Les Quatre Cent Coups	F. Truffaut	J.P. Leaud
	Metropolis	F. Lang	B. Helm
	Chimes at Midnight	O. Welles	J. Moreau

Lieu

Cinema	Adresse	No-Telephone
Rex	Bd Poissonniere	42 36 83 93
Champo	R. des Ecoles	43 54 51 60
Cinoche	R. de Conde	46 33 10 82

Pariscope

Cinema	Titre	Heure
Rex	Jules et Jim	18
Rex	Jules et Jim	20
Cinoche	Jules et Jim	20
Champo	Metropolis	18

Exemple de Requête

Requête: **Qui a dirigé le film Metropolis?**

- On cherche la *valeur* de *l'attribut* `Metteur_en_Scene` de tous les *n-uplets* dans la *relation (table)* `Films` dont *l'attribut* `Titre` est égal à la *constante* `'Metropolis'`.
- **Calcul relationnel n-uplet**

$$\{x.Metteur\_en\_Scene \mid Films(x) \wedge x.Titre =' Metropolis'\}$$

- On cherche `x.Metteur_en_Scene` de tous les *n-uplets* x dans la *relation (table)* `Films` tel que `x.Titre = 'Metropolis'`.

Syntaxe

Symboles (c.a.d. l'alphabet du langage)

Une requête du calcul relationnel est composée de différents symboles :

- Constantes: 'F. Truffaut', 'Jules et Jim', 18, 43 54 51 60,...
- Prédicats (relations): *Movies*,*Location*, ...
- Variables (n-uplets, attributs): x, y, z, ...
- Comparateurs arithmétiques: $=$, $<$, $>$, $<=$, ...
- Connecteurs logiques : $\vee$ (ou), $\wedge$ (et), $\neg$ (non)
- Quantificateurs : $\exists$ (il existe), $\forall$ (pour tous)
- Parenthèses : "(", ")"

Formules atomiques (atomes) et variables libres

Le calcul relationnel est fondé sur la logique du premier ordre et utilise des formules logiques. Les formules les plus simples sont appelés *atomes* :

- p(x1,. . . ,xn) est un *atome* où p est un symbole de prédicat, et xi, i $\in$ [1,n] est soit une variable, soit une constante.

  Exemples :

  - Films (’Jules et Jim’, ’F.Truffaut’, ’J. Moreau’)
  - Films (x, ’Truffaut’, y)
- xθy est un *atome* où x et y sont soit des variables soit des constantes et θ est l’un des 6 comparateurs arithmétiques.

  Exemples : 1 <3, x <y, x <8
- Toutes les occurrences de variables apparaissant dans une formule atomique sont dites **libres**.

Formules bien formées

- si F est une formule avec x parmi ses variables libres, alors (∃x)F et (∀x)F sont des formules. Toutes les occurrences de x dans F sont alors dites *liées*.
  Exemples:
  - $(\exists x)(x < 3)$
  - $(\forall x)(x < 3)$
  - $(\exists x)(x < y)$
- si F1 et F2 sont des formules, alors F1 ∨ F2, F1 ∧ F2 et ¬ F1 sont des formules (les occurrences de variables de ces nouvelles formules sont libres ou liées si elles le sont dans F1, F2). Exemples:
  - $(x <= 3) \vee (x > 5)$
  - $(\exists x)(x < 3 \wedge x > 5)$
- si F est une formule, alors (F) est une formule;

Requêtes

Requête : $\{x1,x2,\ldots,xn \mid F\}$ où F est une formule avec *variables libres*, x1,x2,. . . ,xn.

Exemples de Requêtes :

- $\{x \mid x < 3\}$
- $\{x \mid Film('Jules\ et\ Jim',' Truffaut',x)\}$
- $\{x \mid (\exists y) Film('Metropolis',x,y)\}$

Formes Abrégés

– Formes abrégées de quantificateurs:
  1. $\exists x1,x2,\ldots,xn$ est une abréviation de $\exists x1 \exists x2 \ldots \exists xn$,
  2. $\forall x1,x2,\ldots,xn$ est une abréviation de $\forall x1 \forall x2 \ldots \forall xn$.
– Ordre de précédence (priorité) entre les connecteurs et quantificateurs (du plus au moins prioritaire):
  1. $\neg,\forall,\exists$,
  2. $\wedge$,
  3. $\vee$.

  Par exemple, $(\forall x)\neg p(x,y) \vee q(y) \wedge r(z)$ est compris comme

$$((\forall x)(\neg p(x,y))) \vee (q(y) \wedge r(z)).$$

Sémantique

Interprétation: Exemple

Les formules et requêtes sont “interprétées” :

- Formule: $Film('Metropolis','F.Lang','M.Brando')\}$
  - Interpretation “naturelle”: “Est-ce que M. Brando a joué dans un film Metropolis dirigé par un certain F. Lang?”
  - Interpretation base de données: “Est-ce qu’il existe un n’uplet (’Metropolis’, ’F. Lang’, ’M. Brando’) dans la table $Film$?”

- Formule: $(\exists y) Film('Metropolis','F.Lang',y)\}$
  - Interpretation “naturelle”: “Est-ce qu’il existe un acteur qui a joué dans un film Metropolis dirigé par un certain F. Lang?” ou “Est-ce que F. Lang à dirigé le film Metropolis (avec au moins un acteur)?”
  - Interpretation base de données: “Est-ce qu’il existe une constante c tel que (’Metropolis’, ’F. Lang’, c) est un n’uplet dans la table $Film$?

- Requête: $\{y \mid (\exists x) Film('Metropolis', x, y)\}$
  - Interpretation “naturelle”: “je cherche le nom de tous les acteurs dans le(s) film(s) avec le titre Metropolis” (je ne m’intéresse pas pariculièrement au(x) metteur(s) en scène).
  - Interpretation “base de données”: “je cherche toutes les constantes c tel qu’il existe pour chacune au moins une constante d tel que (’Metropolis’, d, c) est un n-uplet dans la table Film.

Interprétation: atomes avec constantes

Les formules sont interprêtes par rapport à une base de données b. Le résultat (la réponse) de l'interprétation d'une formule est *vrai* ou *faux* :

- L'atome p(x1,. . . ,xn) où p est un symbole de prédicat (nom de relation) et où les xi i $\in$ [1,n] sont des *constantes*, est *vrai* dans b (b satisfait p(x1,. . . ,xn)) si (x1,. . . ,xn) est un n-uplet de la relation p dans b.
  Exemple : $Film('Jules\ et\ Jim','Truffaut','Moreau')$ est vrai ssi $('Jules\ et\ Jim','Truffaut','Moreau')$ est un n-uplet dans la table $Film$.
- L'interprétation de xθy est naturelle (indépendante de de la base de données).
  Exemple: $3 < 5$ est vrai ssi 3 est inférieur à 5.

Interprétation: variables libres et substitution

- Substitution: Si x est une variable libre dans une formule F on note $F[x/c]$ la formule qu'on obtient par un remplacement de x par c.
- Exemple:
  - Formule : F = Film('Metropolis', x, y)
  - Substitution de variables libres :
    1. $F[x/'F.Lang']$ = Film('Metropolis', 'F. Lang', y)
    2. $F[x/'A.Hitchcock']$ = Film('Metropolis', 'A. Hitchcock', y)
    3. $F[x/'F.Lang'][y/'B.Helm']$ = Film('Metropolis', 'F. Lang', 'B. Helm')
    4. $F[y/'B.Helm'][x/'F.Lang']$ = Film('Metropolis', 'F. Lang', 'B. Helm')

Interprétation: quantificateurs

- La formule (∃x)F est *vraie* dans b s'il existe une substition $[x/c]$ de x telle que $F[x/c]$ est *vraie* dans b.
  Par exemple $(\exists x)Films(x,'Truffaut','Moreau')$ est vraie parce que $Films(x,'Truffaut','Moreau')[x/'JulesetJim'] = Films('JulesetJim','Truffaut','Moreau')$ est vraie.
- L'interprétation de F1∨F2. ¬F,... est l'interprétation habituelle : F1∨F2 est vraie si F1 est vraie *ou* F2 est vraie.

Note: La disjonction et la quantification universelle peuvent être exprimées en utilisant la negation:

1. F1 ∨ F2 ≡ ¬(¬ F1 ∧¬ F2),
2. (∀ x) F ≡ (¬∃ x) ¬ F.

Interprétation d'une Requête

Requête : $\{x1,x2,\ldots,xn \mid F\}$ où F est une formule avec les variables libres $x1,x2,\ldots,xn$.

Le résultat de l'interprétation d'une requête est ensemble des n-uplets (une relation!) $[a1,a2,\ldots,an]$ tels que $F/[x1/a1,x2/a2,\ldots,xn/an]$ est vraie.

- $\{x \mid Film('Jules\ et\ Jim','Truffaut',x)\}$ retourne tous les acteurs du film 'Jules et Jim' de Truffaut : $\{['J.Moreau'],['O.Werner]\}$.
- $\{x \mid (\exists y)Film(y,'Truffaut',x)\}$ retourne tous les acteurs qui ont tourné avec Truffaut.

Algèbre relationnelle vs Calcul relationnel sain

L'algèbre relationnelle et le calcul relationnel sain ont le même pouvoir d'expression.

Théorême 1 : Toute requête exprimable dans l'algèbre relationnelle est exprimable dans le calcul relationnel.

Formule saine : Formule dont le résultat est fini (le nb de n-uplets qui satisfont la formule est fini).

Exemple de formule non saine : $\{x|\neg p(x)\}$. Le resultat contient tous les nuplets qui ne sont pas dans la relation p.

Deux solutions : interdire les formules non-saine ou interprétation par rapport au domaine actif de la base de données (exemple : toutes les constantes dans b qui ne sont pas dans p).

Théorême 2 : Toute requête du calcul relationnel sain est exprimable en algèbre relationnelle.

Calcul relationnel domaine/n-uplet

Le calcul relationnel précédent est appelé **calcul relationnel domaine**.

Pour obtenir le **calcul relationnel n-uplet**, on remplace :

- x1,x2,. . . ,xn (où xi est une variable domaine) par la variable n-uplet t.
- L'attribut A de rang i (xi dans le calcul domaine) est noté t.A (voir exemples ci-dessous).

Theorême 3 : Le calcul relationnel n-uplet sain a le même pouvoir d'expression que l'algèbre relationnelle.

Exemples de Requêtes

Quelques Requêtes

1. Qui dirige Metropolis?
2. Adresse et numéro de téléphone du Studio?
3. Salles où on peut voir un film de Truffaut?
4. Adresses des cinémas montrant un film de Truffaut?
5. Quels films de Truffaut ne passent pas en ce moment?

Requête 1: Qui a dirigé le film Metropolis?

SQL

```
select Metteur-en-Scene
  from Films
 where Titre = 'Metropolis';
```

Calcul relationnel n-uplet

$$\{x.Metteur\_en\_Scene \mid Films(x) \wedge x.Titre =' Metropolis'\}$$

Calcul relationnel domaine

$$\{d \mid (\exists x) Films('Metropolis', d, x)\}$$

Requête 2: Adresse et numéro de téléphone du Studio ?

Calcul relationnel n-uplet

$$\{x.Adresse,x.Numero\_Tel \mid Lieu(x) \wedge x.Cinema =' Studio'\}$$

Calcul relationnel domaine

$$\{ad,ph \mid Lieu('Studio',ad,ph)\}$$

Requête 3: Salles où on peut voir un film de Truffaut?

Calcul relationnel n-uplet

$$\{x.Cinema \mid (\exists y)(Pariscope(x) \wedge Films(y) \wedge x.Titre = y.Titre \wedge y.Metteur\_en\_Scene =' Truffaut')\}$$

Calcul relationnel domaine

$$\{s \mid (\exists hor,act,titre)(Pariscope(s,titre,hor) \wedge Films(titre,'Truffaut',act))\}$$

Requête 4: Adresses des cinémas montrant un Truffaut ?

SQL

```
select L.Adresse
  from Films F, Lieu L, Pariscope P
 where F.Metteur-en-Scene = 'Truffaut'
   and P.Titre = F.Titre
   and L.Cinema = P.Cinema;
```

Calcul relationnel n-uplet

$$\{z.Adresse \mid (\exists x,y)(Films(x) \wedge Pariscope(y) \wedge Lieu(z) \wedge x.Metteur\_en\_Scene =' Truffaut' \wedge x.Titre = y.Titre \wedge y.Cinema = z.Cinema)\}$$

Requête 4: Adresses des cinémas montrant un Truffaut ?

Calcul relationnel domaine

$$\{a \mid \quad (\exists t,cine,s,teleph,act)(Films(t,\text{“}Truffaut\text{”},act) \wedge$$
$$Pariscope(cine,t,s) \wedge Lieu(cine,a,teleph))\}$$

Requête 5: Quels films de Truffaut ne passent pas?

SQL

```
select Titre
  from Films F
 where F.Metteur en Scene = 'Truffaut'
   and not exists ( select *
                      from Pariscope P
                     where  P.Titre = F.Titre )
```

Calcul relationnel n-uplet

$$\{x.Titre \mid (Films(x) \wedge x.Metteur\_en\_Scene =' Truffaut') \wedge (\neg\exists y)(Pariscope(y) \wedge y.Titre = x.Titre)\}$$

Requête 5: Quels films de Truffaut ne passent pas?

Calcul relationnel domaine

$$\{x \mid (\exists y) Films(x, \text{“}Truffaut\text{”}, y) \wedge (\neg \exists z, w) Pariscope(z, x, w)\}$$

ou

$$\{x \mid (\exists y) Films(x, \text{“}Truffaut\text{”}, y) \wedge (\forall z, w) \neg Pariscope(z, x, w)\}$$

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).

SQL91 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 $=$, $<$, $>$, $<>$, . . . ,

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, IS NULL, 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.

La valeur NULL (*)

La valeur NULL est une valeur “spéciale” qui représente une *valeur (information) inconnue*.

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 $+$, $-$, $*$,$/$).

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.

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 :

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

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 SECONI
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
```

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`

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 ?

Jointures dans SQL2 (*)

Opérations de Jointure (*)

SQL2	opération	**Algèbre**
R1 CROSS JOIN R2	produit cartesien	$R1 \times R2$
R1 JOIN R2 ON R1.A <R2.B	théta-jointure	$R1 \bowtie_{R1.A<R2.B} R2$
R1 NATURAL JOIN R2	jointure naturelle	$R1 \bowtie R2$

Jointure Naturelle: Exemple (*)

SCHEMA: **EMPLOYE**(EMPNO,ENOM,DEPNO,SAL), **DEPT**(DEPNO,DNOM)

REQUÊTE: *Noms des départements avec les noms de leurs employés.*

SQL:

```
SELECT DNOM, ENOM
FROM   DEPT NATURAL JOIN EMP
```

Note : Comme dans la définition algébrique, l'expression `DEPT NATURAL JOIN EMP` fait la jointure naturelle (sur l'attribut `DEPNO`) et l'attribut `DEPNO` n'apparaît qu'une seule fois dans le schéma du résultat.

Theta-Jointure: Exemple (*)

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

REQUÊTE: *Nom et salaire des employés gagnant plus que l'employé 12546*

SQL:

```
SELECT E1.ENOM, E1.SAL
FROM   EMPLOYE E1 JOIN EMPLOYE E2 ON E1.SAL > E2.SA
WHERE  E2.EMPNO = 12546
```

Jointure Externe (*)

EMP

EMPNO	*DEPNO*	SAL
Tom	1	10000
Jim	2	20000
Karin	3	15000

DEPT

DEPNO	DNOM
1	Comm.
2	Adm.
4	Tech.

Jointure : les n-uplets qui ne peuvent pas être joints sont éliminés :

EMP NATURAL JOIN DEPT

Tom	1	10000	Comm.
Jim	2	20000	Adm.

Jointure Externe (*)

Jointure externe : les n-uplets qui ne peuvent pas être joints *ne sont pas éliminés*.

– On garde tous les n-uplets des deux rélations :

EMP NATURAL *FULL* OUTER JOIN DEPT

Tom	1	10000	Comm.
Jim	2	20000	Adm.
Karin	3	15000	NULL
NULL	4	NULL	Tech.

– On garde tous les n-uplets de la première rélation (gauche) :

EMP NATURAL *LEFT* OUTER JOIN DEPT

Tom	1	10000	Comm.
Jim	2	20000	Adm.
Karin	3	15000	NULL

– On garde tous les n-uplets de la deuxième rélation (droite) :

EMP NATURAL *RIGHT* OUTER JOIN DEPT

Tom	1	10000	Comm.
Jim	2	20000	Adm.
NULL	4	NULL	Tech.

Jointures Externes dans SQL2 (*)

- R1 NATURAL FULL OUTER JOIN R2 : Remplir R1.* et R2.*
- R1 NATURAL LEFT OUTER JOIN R2 : Remplir R2.*
- R1 NATURAL RIGHT OUTER JOIN R2 : Remplir R1.*

avec NULL quand nécessaire.

D’une manière similaire on peut définir des théta-jointures externes :

- R1 (FULL|LEFT|RIGHT) OUTER JOIN R2 ON prédicat

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 est 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 : le résultat contient chaque n-uplet $a - b$ fois, où a et b est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

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 est 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 $<$, $=$, $>$, . . .

- 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(PRIX
  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. Par exemple on garde les produits dont le nombre des fournisseurs est $>= 2$. De cette façon des conditions de sélection peuvent être appliquées avant le calcul d'agrégat (clause WHERE) mais aussi après (clause HAVING).

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).

Récursion dans SQL3 (*)

Enfants (*)

Soit le schéma de relation **ENFANT**(NOMPAR,NOMENF)

REQUÊTE: *Les enfants de Charlemagne*

SQL:

```
SELECT NOMENF
  FROM ENFANT
 WHERE NOMPAR='Charlemagne';
```

Descendants (*)

Soit le schéma de relation **ENFANT**(NOMPAR,NOMENF).

REQUÊTE: *Les* **descendants** *de Charlemagne*

SQL:

```
WITH RECURSIVE DESCENDANT(NOMANC, NOMDESC) AS
      (SELECT NOMPAR, NOMENF FROM ENFANT)
  UNION
      (SELECT R1.NOMANC, R2.NOMDESC
         FROM DESCENDANT R1, DESCENDANT R2
        WHERE R1.NOMDESC=R2.NOMANC)
SELECT NOMDESC FROM DESCENDANT
 WHERE NOMANC='Charlemagne';
```

Mises à jour avec SQL

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)
```

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'*