Bases de Données

Alain Griffault
Université Bordeaux I
Version $Id: bd.tex,v 1.7 2005/12/09 10:40:14 griffaul Exp $

12 décembre 2005

Index

Cours 1 11
Cours 2 16
Cours 3 20
Cours 4 23
Cours 5 28
Cours 6 33
Cours 7 36
Cours 8 38
Cours 9 43
Cours 10 46
Cours 11 51
Cours 12 53

Table des matières

1 Introduction **11**
1.1 Historique . . . 11
1.2 Le traitement d'un programme ou d'une requête . . . 12
1.2.1 Un exemple : l'âge d'une personne . . . 12
1.3 Les problèmes . . . 12
1.3.1 Le niveau physique . . . 12
1.3.2 L'évolution . . . 12
1.4 Le contenu du cours . . . 13
1.5 Les contrôles . . . 13
1.6 Bibliographie . . . 13

2 Le modèle relationnel **15**
2.1 Un concept unique : la relation . . . 15
2.1.1 Un exemple . . . 15
2.2 Ajout de sémantique dans le concept de relation . . . 16
2.3 Un problème classique . . . 17
2.3.1 Première solution . . . 17
2.3.2 Deuxième solution . . . 17
2.3.3 Troisième solution . . . 17
2.4 Le modèle conceptuel relationnel . . . 18

3 L'algèbre relationnelle **19**
3.1 Les opérateurs unaires . . . 19
3.1.1 La sélection (notée $\sigma[\,]$) . . . 19
3.1.2 La projection (notée $\pi[\,]$) . . . 19
3.1.3 Le renommage (noté $\alpha[\,]$) . . . 20
3.2 Les opérateurs binaires . . . 20
3.2.1 L'union (notée $\cup$) . . . 20
3.2.2 La différence (notée $-$) . . . 21
3.2.3 Le produit cartésien (noté $\times$) . . . 21
3.3 Les opérateurs déduits . . . 22
3.3.1 L'intersection (notée $\cap$) . . . 22
3.3.2 La jointure naturelle (notée $\bowtie$) . . . 22
3.3.3 La thêta-jointure (notée $\theta[\,]$) . . . 22
3.3.4 La division (notée $/$) . . . 23
3.4 Quelques propriétés . . . 23
3.5 Optimisation des calculs . . . 23

4 SQL : un langage de description de schémas relationnels 25
4.1 Description de schémas conceptuels . . . 25
4.1.1 Création de tables . . . 25
4.1.2 Déclaration de clés . . . 25
4.1.3 Déclaration de contraintes . . . 25
4.1.4 L'exemple des étudiants . . . 26
4.1.5 Modification du contenu d'une table . . . 26
4.2 Description de schémas externes . . . 26
4.2.1 L'exemple des étudiants . . . 26

5 SQL : un langage algébrique de manipulation de données 27
5.1 Forme générale d'une requête SQL . . . 27
5.2 Les opérateurs unaires . . . 27
5.2.1 La sélection . . . 27
5.2.2 La projection . . . 28
5.2.3 Le renommage . . . 28
5.3 Les opérateurs binaires . . . 28
5.3.1 L'union . . . 28
5.3.2 La différence . . . 29
5.3.3 Le produit cartésien . . . 29
5.4 Les requêtes imbriquées . . . 30
5.5 Les opérateurs déduits . . . 30
5.5.1 L'intersection . . . 30
5.5.2 La jointure naturelle . . . 30
5.5.3 La thêta-jointure . . . 31
5.5.4 La division . . . 31

6 Théorie de la normalisation 33
6.1 Quel schéma conceptuel choisir ? . . . 33
6.1.1 Un exemple . . . 33
6.1.2 Les problèmes potentiels d'un schéma conceptuel . . . 34
6.2 Dépendances fonctionnelles et décomposition . . . 34
6.3 Graphes des dépendances fonctionnelles d'une relation . . . 35
6.3.1 Exemple . . . 36
6.4 Formes normales . . . 36
6.4.1 Première forme normale (1NF) . . . 36
6.4.2 Seconde forme normale (2NF) . . . 36
6.4.3 Troisième forme normale (3NF) . . . 37
6.4.4 Forme normale de Boyce-Codd (BCNF) . . . 37
6.4.5 Remarques . . . 38
6.4.6 Perte de dépendances fonctionnelles . . . 38
6.5 Algorithmes de normalisation . . . 38
6.5.1 Décomposition en 3NF . . . 38
6.5.2 Décomposition en BCNF . . . 40
6.6 Intérêt de la normalisation . . . 41
6.7 Modèle conceptuel Entité-Association et normalisation . . . 41

7 Les calculs relationnels 43
7.1 Introduction . . . 43
7.2 Le calcul de tuples . . . 43
7.2.1 Exemples . . . 43
7.2.2 Remarques . . . 44
7.3 Le calcul de domaines . . . 44
7.3.1 Exemples . . . 44

7.3.2 Remarques 44

8 SQL : un langage prédicatif 45
8.1 Forme générale d'une requête SQL emboitée 45
8.2 L'appartenance à un ensemble 45
8.3 Les quantifications 45
8.4 Les conditions sur des ensembles 46
8.4.1 Les comparaisons entre ensembles 46
8.4.2 L'existence d'un élément 47
8.5 Et la division ? 47

9 SQL : un langage de présentation des données 49
9.1 Les fonctions d'agrégation 49
9.2 Partitionnement des tuples d'une relation 49
9.3 L'ordonnancement des tuples 50
9.4 Autres fonctionnalités 50

10 Gestion de la correction : les transactions 51
10.1 Cohérence et correction 51
10.1.1 Un exemple 51
10.2 La transaction : une unité logique de travail 52
10.2.1 Les requêtes sont des transactions 52
10.3 La gestion des transactions 52
10.3.1 Journal et points de validation 52
10.3.2 La mise en œuvre d'une transaction 52
10.3.3 La reprise après une panne 53

11 La gestion des accès concurrents 55
11.1 Les problèmes potentiels 55
11.1.1 Les dépendances non valides dues à un échec 55
11.1.2 Les états incorrects 55
11.2 La sérialisation 56
11.3 Une solution : Les verrous 56
11.3.1 Le verrouillage strict à deux phases 56
11.3.2 Exemples sur les problèmes 56
11.4 Les blocages 57
11.5 Des solutions aux problèmes du blocage 57
11.5.1 Détection et correction 57
11.5.2 Évitement 58
11.6 Et en pratique 58
11.6.1 Du point de vue des SGBD 58
11.6.2 Du point de vue SQL 58
11.6.3 Du point de vue exploitation 58

Table des figures

1.1 L'architecture classique d'un SGBD 11
1.2 Les étapes du traitement d'une requête 12

Liste des tableaux

2.1 La relation étudiants de Bordeaux 1 15
2.2 La relation listeDePrenoms de Bordeaux 1 17
2.3 La relation PlusieursPrenoms de Bordeaux 1 17
2.4 Les relations Étudiants et AutresPrénoms de Bordeaux 1 17

3.1 L'opération $\sigma[(\texttt{Nom} = \texttt{Nom2})|(\texttt{Datedenaissance} = \texttt{31mars1985})](\texttt{Etudiants})$ 19
3.2 L'opération $\pi[(\texttt{Nom}, \texttt{Prenom})](\texttt{Etudiants})$ 20
3.3 L'opération $\alpha[(\texttt{Datedenaissance} : \texttt{Ne(e)le})](\texttt{Etudiants})$ 20
3.4 L'union de deux relations 21
3.5 La différence de deux relations 21
3.6 Le produit cartésien de deux relations 21
3.7 L'intersection de deux relations 22
3.8 La jointure naturelle de deux relations 22
3.9 La thêta-jointure de deux relations 22
3.10 La division de deux relations 23

Chapitre 1

Introduction

1.1 Historique

Historiquement, les B.D. sont apparues dans le but de répondre à plusieurs problèmes :
- Limiter le nombre d'interfaces utilisateurs.
- Diminuer les dépendances entre les données et les programmes d'application ;
- Diminuer les redondances d'informations, afin d'une part de diminuer la place nécessaire au stockage ; d'autre part de diminuer les incohérences et les temps de mise à jour.

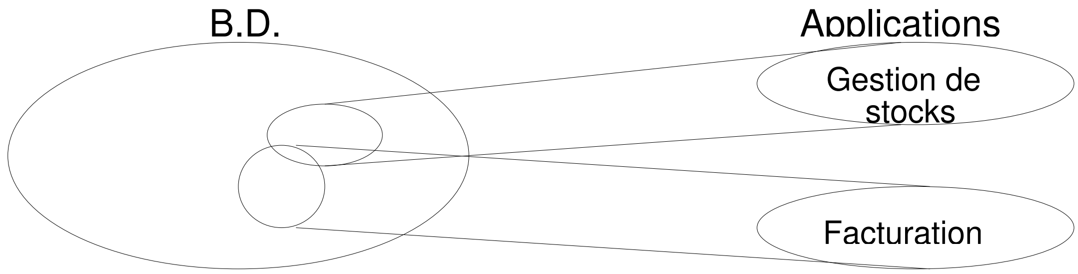

Fig. 1.1 – L'architecture classique d'un SGBD

B.D. : C'est un ensemble de données (sans redondance si possible) ou les liaisons entre les entités sont conservées ; disponible simultanément pour plusieurs applications ; et géré par un S.G.B.D.

S.G.B.D. : C'est d'une part un ensemble de modules qui assurent l'interface entre les usagers et les informations ; d'autre part un outil qui permet la gestion d'informations (recherche, insertion, suppression, modification). Ses principales caractéristiques (confidentialité, partage des ressources, accès concurrent et sécurité de fonctionnement) font qu'il est généralement partitionné fonctionnellement en trois parties :

interne Ce sont les modules qui gèrent les liens entre les données manipulées, et les structures qui les stockent. Ils contiennent notamment toute l'algorithmique pour la gestion des informations.

externe Ce sont les modules qui produisent les images des données, qui sont utilisables par les programmes d'applications, et par les utilisateurs. Ils assurent les différents interfaces utilisateurs.

conceptuel/logique Ce sont des modules qui gèrent des fichiers d'informations propres au S.G.B.D. ainsi que des fichiers qui contiennent les schémas conceptuels des données.

1.2 Le traitement d'un programme ou d'une requête

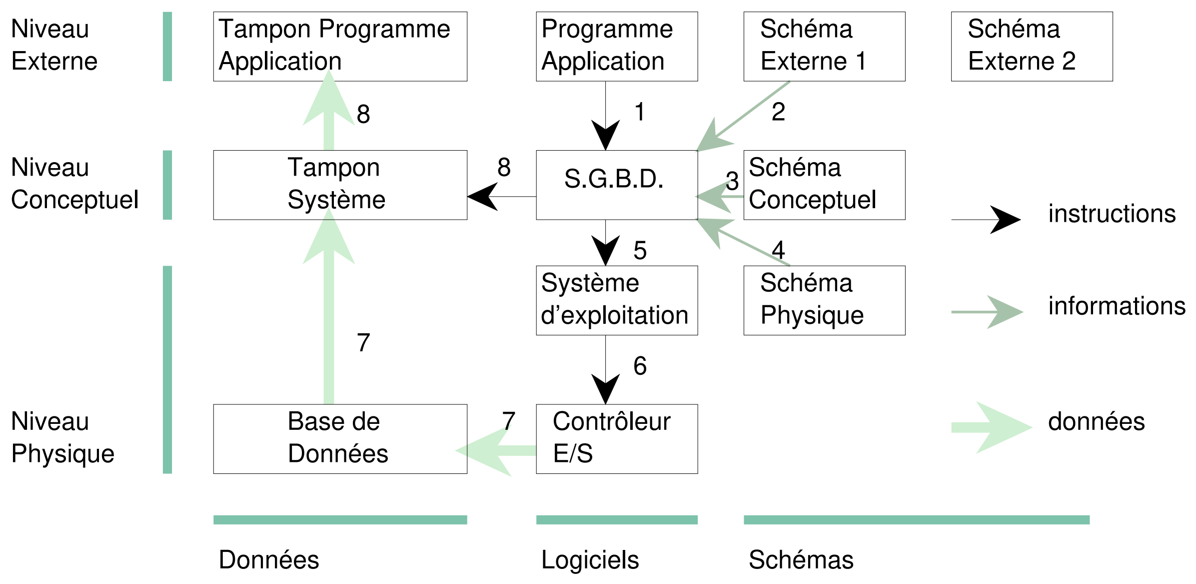

FIG. 1.2 – Les étapes du traitement d'une requête

1.2.1 Un exemple : l'âge d'une personne

Externe : Âge de Zinedine Zidane : 33 ans.

Conceptuel : Date de naissance de Zinedine : 23 juin 1972.

Physique : Plusieurs choix :
- trois entiers (23, 6, 1972).
- le nombre de jours entre naissance et une date (1,1,1900) : 26076.

1.3 Les problèmes

1.3.1 Le niveau physique

Le temps d'accès moyen à un disque (10 ms) est très supérieur au temps d'accès à la RAM. L'optimisation de la couche physique consiste donc à diminuer le nombre d'accès au disque. Les solutions concernent l'organisation et la compression des données.
Remarques :
- Pas de solution optimale pour tout type de problèmes ;
- Un bon S.G.B.D. permet une grande variété de structures de stockage.
- Pour une grande base de données (>> taille de la RAM), l'organisation des données est une tâche difficile, et dépend beaucoup de l'utilisation qui en est faîte.

1.3.2 L'évolution

	Volume en octets	Modèle
Avant 1985	Mega	Hiérarchique, Transactionnel, Gros serveur
1995	Giga	Relationnel, Client-serveur
2005	Tera	Entrepôt de données
...	Exa (10^{18})	Challenge

1.4 Le contenu du cours

Le cours traite des B.D les plus utilisées aujourd'hui : les bases de données relationnelles qui sont les grandes bases d'hier, et les bases de la très grande majorité des entreprises actuelles.

Étape 1 : nécessite un langage de manipulation de données (LMD).
- Langage de requêtes OUI
- Langage de programmation incluant des requêtes. NON
- Langage de requêtes avec des structures de contrôles. NON

Étape 2 : nécessite un langage de description de liens entre les données applications et les données logiques. OUI

Étape 3 : nécessite un langage de description de données (LMD).

Étape 4 : nécessite un langage de description de liens entre données logiques et d'organisation des structures de données dans le système de fichiers. NON

Étape 5 : algorithmique optimisée pour demander au SE les fichiers (ou directement les pages) contenant les données. OUI-NON

Étape 6 : gestionnaire de disques. NON

Étape 8 : Mise en forme du résultat. NON

1.5 Les contrôles

Un projet et un examen.

1.6 Bibliographie

- `http://lbdwww.epfl.ch/teaching/courses/poly1`
- *An Introduction to Database Systems, Eighth Edition* C.J. Date.

Chapitre 2

Le modèle relationnel

Le modèle relationnel est du à Codd en 1960. Il a été l'objet de nombreuses recherches jusque dans les années 1990. C'est le modèle le plus utilisé par les SGBDs actuellement disponibles sur le marché.

C'est un modèle très simple. Trop simple pour certains qui ont proposé d'autres modèles conceptuels permettant de mieux décrire certains aspects *sémantique* des données : modèles Entité-Association, relationnel-objet...

2.1 Un concept unique : la relation

2.1.1 Un exemple

Chaque étudiant de l'Université Bordeaux 1 est décrit par plusieurs attributs.

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

TAB. 2.1 – La relation étudiants de Bordeaux 1

Définition 2.1 (Domaine) *Un domaine D est un ensemble de valeurs atomiques (non décomposable) que peut prendre un attribut.*

Quelques exemples de domaines :
- les chaînes de caractères de longueur maximale 30.
- les entiers positifs.
- les entiers compris entre 1 et 12.
- les couleurs {Rouge, Vert, Bleu}.
- ...

Définition 2.2 (n-uplet) *Soit* $\{D_1, D_2, \ldots, D_n\}$ *une liste de domaines. Un n-uplet ou tuple est un vecteur de valeurs* $\{v_1 \in D_1, v_2 \in D_2, \ldots, v_n \in D_n\}$.

Définition 2.3 (Relation) *Soit* $\{D_1, D_2, \ldots, D_n\}$ *une liste de domaines. Une relation R est un ensemble de n-uplets* $\{v_1 \in D_1, v_2 \in D_2, \ldots, v_n \in D_n\}$.

Remarque : Une relation est un ensemble et non un multi-ensemble. Chaque n-uplet est donc unique, et il n'y a pas d'ordre entre les n-uplets.

$$\boxed{\begin{matrix} a & 3 \\ b & 8 \\ f & 5 \end{matrix}} = \boxed{\begin{matrix} a & 3 \\ b & 8 \\ f & 5 \\ b & 8 \end{matrix}} = \boxed{\begin{matrix} b & 8 \\ a & 3 \\ f & 5 \end{matrix}}$$

Remarque : d'un point de vue ensembliste, $R \subseteq \prod_{i=1}^{i=n} D_i$.

2.2 Ajout de sémantique dans le concept de relation

Définition 2.4 (Identifiant ou clef) *Soit une liste d'attributs $A_1, A_2, \ldots, A_n$ sur une liste de domaine $\{D_1, D_2, \ldots, D_n\}$, et une relation $R \subseteq \prod_{i=1}^{i=n} D_i$.*

L'ensemble $I = \{I_1, I_2, \ldots, I_m\} \subseteq \{A_1, A_2, \ldots, A_n\}$ est un identifiant si et seulement si :
- *il n'existe pas deux n-uplets dans R ayant même valeur sur l'ensemble I.*
- *pour tout sous-ensemble $J \subset I$, il existe deux n-uplets dans R ayant même valeur sur l'ensemble J.*

Remarque : Pour la relation *Étudiants* de la table 2.1, le seul identifiant possible est *Numéro*.
Remarque : Si la relation *R évolue* au cours du temps, les ensembles identifiant peuvent évoluer aussi.
Soit le scénario suivant sur la relation *Étudiants* de la table 2.1 :

1. retrait du n-uplet (325, nom1, prénom1, 16 juin 1985). Les ensembles {Numéro} et {Date de naissance} sont des identifiants.
2. ajout du n-uplet (65758, nom4, prénom4, 31 mars 1985). L'ensemble de tous les attributs est l'identifiant.

Ce scénario n'est pas souhaitable. L'identifiant d'une relation n'est donc pas un ensemble calculé dynamiquement, mais une notion statique donnée par le concepteur de la relation.

Cours 2

Définition 2.5 (Contraintes d'intégrité) *Soit une liste d'attributs $A_1, A_2, \ldots, A_n$ sur une liste de domaine $\{D_1, D_2, \ldots, D_n\}$, et une relation $R \subseteq \prod_{i=1}^{i=n} D_i$.*

Soit P une propriété logique sur la liste d'attributs. P est une contrainte d'intégrité pour la relation R si et seulement si :
- *P restreint à R est vrai.*
- *Toute modification de R laisse P vrai.*

Il est possible de classer les contraintes d'intégrité :

Élémentaire Pour chaque n-uplet v de la relation, il est possible de décider si $P(v)$ est vrai ou faux.

Statique Pour chaque n-uplet v de la relation, il n'est pas possible de décider si $P(v)$ est vrai ou faux, mais il est possible de décider si $P(R)$ est vrai ou faux.

Dynamique Il n'est pas possible de décider si $P(R)$ est vrai ou faux, mais il est possible de décider si $P(R \longrightarrow R)$ est vrai ou faux.

Exemples :

Élémentaire $P1$: les numéros des étudiants ne sont pas des multiples de 3. Le prédicat $P1$ interdit l'ajout du n-uplet (369, nom4, prénom4, 31 mars 1985).

Élémentaire $P2$: la chaîne de caractère de l'attribut *Date de naissance* est une date valide. Le prédicat $P2$ interdit l'ajout du n-uplet (369, nom4, prénom4, 29 février 2100).

Statique $P3$: le nombre de *Numéro* finissant par 83 est inférieur ou égal à 17.

Statique $P4 = P1|(P2\&P3)$.

Dynamique $P5$: les étudiants né(e)s un 29 février ne peuvent pas être supprimés de la base.

Les contraintes d'intégrité d'une relation sont des propriétés logiques qui restreignent les évolutions de la relation. Elles sont données par le concepteur de la relation. La contrainte $P4$ précédente dit que l'on peut toujours regrouper la liste des contraintes d'intégrité en une seule plus complexe grâce aux opérateurs booléens.

2.3 Un problème classique

On désire mettre la liste ordonnée des prénoms des étudiants dans la base de données.

2.3.1 Première solution

Numéro	Nom	Prénoms	Date de naissance
65758	nom1	prénom1 prénom3 prénom6	16 juin 1985
7897090	nom2	prénom2 prénom5	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1 prénom2	16 juin 1985
45365	nom3	prénom2 prénom4	20 septembre 1986
⋮	⋮	⋮	⋮

Tab. 2.2 – La relation listeDePrenoms de Bordeaux 1

Défaut : Du fait que les attributs sont atomiques, c'est aux applications de faire l'analyse syntaxique de la liste des prénoms afin de n'afficher que le premier lorsque seul le premier est demandé.

2.3.2 Deuxième solution

Numéro	Nom	Prénom1	Prénom2	Prénom3	Date de naissance
65758	nom1	prénom1	prénom3	prénom6	16 juin 1985
7897090	nom2	prénom2	prénom5		29 février 1984
7689689	nom1	prénom3			31 mars 1985
325	nom1	prénom1	prénom2		16 juin 1985
45365	nom3	prénom2	prénom4		20 septembre 1986
⋮	⋮	⋮	⋮	⋮	⋮

Tab. 2.3 – La relation PlusieursPrenoms de Bordeaux 1

Défaut :
- Il faut fixer à priori le nombre maximal de prénoms.
- Que faire si ce nombre n'est pas suffisant ?
  Pas de solution.
- Que mettre comme valeur lorsqu'il n'y a pas assez de prénoms ?
  Ajout d'une valeur spécifique *Non défini*. Mais cette valeur ne peut pas être ajoutée au domaine, car il faut au moins un prénom défini. Bien que non satisfaisante, cette solution est quelquefois retenue.

2.3.3 Troisième solution

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

Numéro	Ordre	Prénom
65758	2	prénom3
65758	3	prénom6
7897090	2	prénom5
325	2	prénom2
45365	2	prénom4
⋮	⋮	⋮

Tab. 2.4 – Les relations Étudiants et AutresPrénoms de Bordeaux 1

Remarque Cette description est satisfaisante si l'on peut spécifier que :
– {Numéro} est l'identifiant de la relation *Étudiants*
– {Numéro, Ordre} est l'identifiant de la relation *AutresPrénoms*
– (num, indice) $\in$ *AutresPrénoms* $\Rightarrow$ indice = 2 ou (num, indice-1) $\in$ *AutresPrénoms*.
– $\forall n \in$ *AutresPrénoms*.Numéro, $\exists m \in$ *Noms*.Numéro tel que $n = m$
Cette solution est sans doute la meilleure solution, car elle évite le problème de la valeur *Non défini*.

2.4 Le modèle conceptuel relationnel

Définition 2.6 (Identifiant externe / Clef externe / Clef étrangère) *Soient deux relations R^1 et R^2 sur les listes d'attributs $(A_1^1, A_2^1, \ldots, A_n^1)$ et $(A_1^2, A_2^2, \ldots, A_m^2)$.*

L'ensemble $I = \{I_1, I_2, \ldots, I_p\} \subseteq \{A_1^1, A_2^1, \ldots, A_n^1\}$ est un identifiant externe de la relation R^1 si et seulement si :
– *I est un identifiant de R^2.*
– $\forall (v_1^1, v_2^1, \ldots, v_n^1) \in R^1, \exists (v_1^2, v_2^2, \ldots, v_m^2) \in R^2$ *tel que les valeurs $v_{I_1}, v_{I_2}, \ldots, v_{I_p}$ coïncident.*

Remarques :
– Cette notion est sémantique et doit être déclarée par le concepteur.
– Le maintien de cette propriété sémantique est complexe.

1. L'ajout du n-uplet (65758, 4, prénom7) ne pose pas de problème.
2. Le retrait du n-uplet (7897090, 2, prénom5) ne pose pas de problème.
3. Le retrait du n-uplet (325, nom1, prénom1, 16 juin 1985) implique le retrait du n-uplet (325, 2, prénom2).

Définition 2.7 (Contraintes d'intégrité inter-relations) *Soient une liste de relations R^i et les listes d'attributs $(A_1^i, A_2^i, \ldots, A_{n_i}^i)$.*

Soit P un prédicat sur l'union des listes d'attributs. P est une contrainte d'intégrité pour les relations R^i si et seulement si :
– *P est vrai.*
– *Toute modification des R^i laisse P vrai.*

Définition 2.8 (Un schéma relationnel) *se compose de :*
– *un ensemble de relations. Pour chacune, obligatoirement :*
  – *un nom,*
  – *une liste d'attributs et la liste des domaines associés,*
  – *un identifiant,*
  et éventuellement :
  – *une liste d'identifiants externes,*
  – *une contrainte d'intégrité.*
– *une éventuelle contrainte d'intégrité inter-relations.*

Chapitre 3

L'algèbre relationnelle

C'est un ensemble d'opérations, qui à partir de relations, construisent d'autres relations.

Les opérateurs unaires sont la sélection (notée $\sigma[\,]$), la projection (notée $\pi[\,]$) et le renommage (noté $\alpha[\,]$).

Les opérateurs binaires sont l'union (noté $\cup$), la différence (notée $-$) et le produit (noté $\times$).

Les opérateurs déduits sont obtenus par composition des précédents. Ils sont pratiques, mais n'apportent aucune expressivité supplémentaire. On utilise généralement l'intersection (notée $\cap$), la jointure (notée $\bowtie$), la thêta-jointure (notée $\theta[\,]$) et la division (notée $/$).

3.1 Les opérateurs unaires

3.1.1 La sélection (notée $\sigma[\,]$)

Définition 3.1 (Selection) *Soit R une relation sur les attributs $(A_1, A_2, \ldots, A_n)$. Soit p un prédicat élémentaire sur les attributs $(A_1, A_2, \ldots, A_n)$.*
$\sigma[p](R) = \{(v_1, v_2, \ldots, v_n) \in R, \ \texttt{et}\ p(v_1, v_2, \ldots, v_n)\}$

La sélection construit la relation constituée des n-uplets de R qui satisfont p. (La sélection supprime des n-uplets).

Exemple

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

→

Numéro	Nom	Prénom	Date de naissance
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
⋮	⋮	⋮	⋮

TAB. 3.1 – L'opération $\sigma[(\texttt{Nom} = \texttt{Nom2}) | (\texttt{Datedenaissance} = \texttt{31mars1985})](\texttt{Etudiants})$

Complexité : Dans le pire des cas, il faut itérer sur tous les n-uplets de la relation, soit $O(|R|)$.

3.1.2 La projection (notée $\pi[\,]$)

Définition 3.2 (Projection) *Soit R une relation sur les attributs $A = \{A_1, A_2, \ldots, A_n\}$.*
Soit $S = \{S_1, S_2, \ldots, S_m\} \subset A$ un sous-ensemble des attributs. Posons $B = A - S = \{B_1, B_2, \ldots, B_{n-m}\}$.
$\pi[S](R) = \{(v_1, v_2, \ldots, v_m), \ \texttt{telque}\ \exists (b_1, b_2, \ldots, b_{n-m})\}\ \texttt{et}\ (v_1, v_2, \ldots, v_m, b_1, b_2, \ldots, b_{n-m}) \in R$

La projection construit la relation obtenue à partir de R en éliminant les attributs non présents dans S.

Exemple

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

→

Nom	Prénom
nom1	prénom1
nom2	prénom2
nom1	prénom3
nom1	prénom1
nom3	prénom2
⋮	⋮

→

Nom	Prénom
nom1	prénom1
nom2	prénom2
nom1	prénom3
nom3	prénom2
⋮	⋮

TAB. 3.2 – L'opération $\pi[(\texttt{Nom}, \texttt{Prenom})](\texttt{Etudiants})$

Complexité : Dans le pire des cas, il faut itérer sur tous les n-uplets de la relation, puis enlever les doublons éventuels ce qui nécessite un tri, soit $O(|R|ln(|R|))$. Si la projection contient l'identifiant, il n'y a pas de doublon et la complexité est $O(|R|)$.

3.1.3 Le renommage (noté $\alpha[\,]$)

Cours 3

Définition 3.3 (Renommage) *Soit R une relation sur les attributs $A = \{A_1, A_2, \ldots, A_n\}$.*
Soit B_i un attribut ayant même domaine que l'attribut A_i.
$\alpha[A_i : B_i](R) = \{(v_1, v_2, \ldots, v_n) \in R$

Le renommage construit la relation obtenue à partir de R en remplaçant certains attributs.

Exemple

Numéro	Nom	Prénom	Date de naissance
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

→

Numéro	Nom	Prénom	Né(e) le
65758	nom1	prénom1	16 juin 1985
7897090	nom2	prénom2	29 février 1984
7689689	nom1	prénom3	31 mars 1985
325	nom1	prénom1	16 juin 1985
45365	nom3	prénom2	20 septembre 1986
⋮	⋮	⋮	⋮

TAB. 3.3 – L'opération $\alpha[(\texttt{Datedenaissance} : \texttt{Ne(e)le})](\texttt{Etudiants})$

3.2 Les opérateurs binaires

3.2.1 L'union (notée $\cup$)

Définition 3.4 (Union) *Soient R^1, R^2 deux relations sur les attributs $(A_1, A_2, \ldots, A_n)$.*
$R^1 \cup R^2 = \{v = (v_1, v_2, \ldots, v_n)$ `telque` $v \in R^1$ `ou` $v \in R^2\}$

L'union construit la relation constituée des n-uplets appartenant soit à R^1 soit à R^2.

Exemple

lettre	chiffre
A	7
F	6
M	7

$\cup$

lettre	chiffre
C	4
F	6
Z	9

$=$

lettre	chiffre
A	7
F	6
M	7
C	4
F	6
Z	9

$=$

lettre	chiffre
A	7
F	6
M	7
C	4
Z	9

TAB. 3.4 – L'union de deux relations

Complexité : Il faut enlever les doublons. Dans le pire des cas, il faut trier les deux relations puis utiliser un algorithme de fusion, soit $O(|R^i|ln(|R^i|)$ avec $|R^i| = max(|R^1|, |R^2|)$.

3.2.2 La différence (notée $-$)

Définition 3.5 (Différence) *Soient R^1, R^2 deux relations sur les attributs $(A_1, A_2, \ldots, A_n)$.*
$R^1 - R^2 = \{v = (v_1, v_2, \ldots, v_n)$ `telque` $v \in R^1$ `et` $v \notin R^2\}$

La différence construit la relation constituée des n-uplets appartenant à R^1 et n'appartenant pas à R^2.
Exemple

lettre	chiffre
A	7
F	6
M	7
C	4

$-$

lettre	chiffre
C	4
F	6
Z	9

$=$

lettre	chiffre
A	7
M	7

TAB. 3.5 – La différence de deux relations

Complexité : Deux possibilités : Pour chaque n-uplet de R^2, il faut l'enlever de R^1, ou bien pour chaque n-uplet de R^1, il faut tester son existence dans R^2. Dans le pire des cas, chacune des possibilités demandent de trier une relation. Il faut donc trier la plus petite des deux, puis utiliser un algorithme de recherche dichotomique. Soit $O(|R^i|ln(|R^i| + |R^j|ln(|R^i|)) = O((|R^1| + |R^2|)ln(|R^i|)$ avec $|R^i| = min(|R^1|, |R^2|)$.

3.2.3 Le produit cartésien (noté $\times$)

Définition 3.6 (Produit) *Soient R^1, R^2 deux relations sur les attributs $(A^1_1, \ldots, A^1_{n_1})$ et $(A^2_1, \ldots, A^2_{n_1})$, tel que $\{A^1_1, \ldots, A^1_{n_1}\} \cap \{A^2_1, \ldots, A^2_{n_1}\} = \emptyset$*
$R^1 \times R^2 = \{(v^1_1, \ldots, v^1_{n_1}, v^2_1, \ldots, v^2_{n_2})$ `telque` $(v^1_1, \ldots, v^1_{n_1}) \in R^1$ `et` $(v^2_1, \ldots, v^2_{n_2}) \in R^2\}$

Le produit construit la relation constituée des n-uplets obtenus en combinant tous les n-uplets de R^1 à tous ceux de R^2.
Exemple

lettre	chiffre
A	7
F	6
C	4

$\times$

indice	pays
33	F
39	I

$=$

lettre	chiffre	indice	pays
A	7	33	F
A	7	39	I
F	6	33	F
F	6	39	I
C	4	33	F
C	4	39	I

TAB. 3.6 – Le produit cartésien de deux relations

Complexité : Pour chaque n-uplet de R^2, il faut itérer R^1, soit $O(|R^1| \times |R^2|)$.

3.3 Les opérateurs déduits

3.3.1 L'intersection (notée $\cap$)

L'intersection construit la relation constituée des n-uplets appartenant à R^1 et à R^2.
Exemple

lettre	chiffre
A	7
F	6
M	7

$\cap$

lettre	chiffre
C	4
F	6
Z	9

$=$

lettre	chiffre
F	6

TAB. 3.7 – L'intersection de deux relations

Définition 3.7 (Intersection) *Soient R^1, R^2 deux relations sur les attributs $(A_1, A_2, \ldots, A_n)$.*
$R^1 \cap R^2 = R^1 - (R^1 - R^2) = R^2 - (R^2 - R^1)$

Complexité : Celle de la différence.

3.3.2 La jointure naturelle (notée $\bowtie$)

La jointure naturelle construit la relation constituée des n-uplets obtenus en combinant tous les n-uplets de R^1 à tous ceux de R^2 ayant les mêmes valeurs sur les attributs communs.
Exemple

lettre	chiffre
A	7
F	6
C	4

$\bowtie$

indice	lettre
33	F
7	F
39	I

$=$

lettre	chiffre	indice
F	6	33
F	6	7

TAB. 3.8 – La jointure naturelle de deux relations

Un renommage pour éviter la confusion, puis un produit, puis la sélection et enfin la projection pour enlever l'attribut renommé devenu inutile car redondant.

Définition 3.8 (Jointure) *Soient R^1 une relation sur les attributs $(A_1, \ldots, A_n, B^1_1, \ldots, B^1_{n_1})$ et R^2 une relation sur $(A_1, \ldots, A_n, B^2_1, \ldots, B^2_{n_2})$.*
$R^1 \bowtie R^2 = \pi[A_i, B^1_j, B^2_k](\sigma[A_i = R^2.A_i](R^1 \times (\alpha[A_i : R^2.A_i](R^2))))$

Complexité : Celle du produit car par construction il n'y a pas de doublon.

3.3.3 La thêta-jointure (notée $\theta[\,]$)

La thêta-jointure construit le produit et ne conserve que les n-uplets qui vérifient P.
Exemple

lettre	chiffre
A	7
F	6
C	4

$\theta[\texttt{lettre} \neq \texttt{pays}]$

indice	pays
33	F
39	I

$=$

lettre	chiffre	indice	pays
A	7	33	F
A	7	39	I
F	6	39	I
C	4	33	F
C	4	39	I

TAB. 3.9 – La thêta-jointure de deux relations

Définition 3.9 (Theta-jointure) *Soient R^1, R^2 deux relations sur les attributs $(A^1_1, \ldots, A^1_{n_1})$ et $(A^2_1, \ldots, A^2_{n_1})$, telles que $\{A^1_1, \ldots, A^1_{n_1}\} \cap \{A^2_1, \ldots, A^2_{n_1}\} = \emptyset$*

Soit P un prédicat élémentaire sur l'union des attributs.

$R^1\theta[P]R^2 = \sigma[P](R^1 \times R^2)$

Complexité : Celle du produit.

3.3.4 La division (notée /)

La division construit la plus grande relation dont le produit avec R^2 est inclus dans R^1. *Cela ressemble à une division entière.*

Exemple

lettre	chiffre	indice	pays
A	7	33	F
A	7	39	I
F	6	39	I
C	4	33	F
C	4	39	I

/

indice	pays
33	F
39	I

=

lettre	chiffre
A	7
C	4

TAB. 3.10 – La division de deux relations

On projette R^1 sur ses attributs propres, puis le produit avec R^2 fourni un sur-ensemble de R^1. La différence avec R^1 représente les n-uplets *non entiers* de R^1, ce sont donc ceux-là qu'ils faut retrancher de la projection de R^1.

Définition 3.10 (Division) *Soient R^1 une relation sur les attributs $(A_1, \ldots, A_n, B_1, \ldots, B_n)$ et R^2 une relation sur $(B_1, \ldots, B_n)$.*

$R^1/\ R^2 = \pi[A_i](R^1) - \pi[A_i]((\pi[A_i](R^1) \times R^2) - R^1)$

Complexité : ?

3.4 Quelques propriétés

Les opérateurs commutatifs et associatifs $\forall$ op $\in \{\cup, \cap, \times, \bowtie, \theta[p]\}$, nous avons R^1 op $R^2 = R^2$ op R^1 et R^1 op $(R^2$ op $R^3) = (R^1$ op $R^2)$ op R^3.

Les compositions d'opérateurs

- $\pi[A^1_i](\pi[A^2_i](\ldots(\pi[A^k_i](R)))) = \pi[\cup_{j=1}^{j=k} A^j_i](R)$
- $\sigma[P^1](\sigma[P^2](\ldots(\sigma[P^k](R)))) = \sigma[\bigwedge_{j=1}^{j=k} P^j](R)$
- $(R^1 \times R^2)/\ R^2 = R^1$
- $(R^1/\ R^2) \times R^2 \subseteq R^1$

Remarques : La différence est ni commutative, ni associative, et le renommage se compose avec *délicatesse.* L'opération de complément n'est pas définie.

3.5 Optimisation des calculs

Cours 4

En utilisant les propriétés des opérateurs et leurs complexités, il est possible par simple réécriture des calculs, d'optimiser le temps nécessaire à l'obtention du résultat.

En réalité, le processus d'optimisation des requêtes est beaucoup plus complexe, car il tient compte également des structures de données utilisées et des tailles des différentes relations.

Chapitre 4

SQL : un langage de description de schémas relationnels

SQL (Structured Query Language) est le langage le plus utilisé aujourd'hui. Il en est à sa troisième version : SQL :92, SQL :99 et SQL :2003.

4.1 Description de schémas conceptuels

4.1.1 Création de tables

```
CREATE TABLE NomDeLaTable (
  NomColonne1 TypeDeDonnée,
  ...
  NomColonneN TypeDeDonnée
);
```

Les types de données dépendent du SGBD. Classiquement, on trouve `CHAR(n)`, `INTEGER`, `FLOAT`, `DATE` ...

4.1.2 Déclaration de clés

```
PRIMARY KEY (NomColonne1, ..., NomColonneK)
```

et pour les clefs étrangères :

```
FOREIGN KEY (NomColonne1, ..., NomColonneK)
REFERENCES NomDeTableExterne(Colonne1, ..., ColonneK)
```

4.1.3 Déclaration de contraintes

Plusieurs mots clefs permettent la définition de contraintes : `ASSERTION`, `CHECK`, `CONSTRAINT`, `DEFAULT`, `NOT NULL`, `UNIQUE`, `TRIGGER`. Par exemple :

```
  DateDeNaissance date      CHECK (DateDeNaissance > 1/1/1900));
```

Les contraintes statiques et dynamiques peuvent être gérées soit par des fonctions SQL, soit par les `TRIGGER`, qui font des appels à des fonctions écrites en `C` ou dans un autre langage.

```
ALTER TABLE NomDeTable ADD CONSTRAINT NomContrainte CHECK condition;
```

```
CREATE TRIGGER NomDuTrigger BEFORE INSERT OR UPDATE OR DELETE ON NomTable
    FOR EACH ROW EXECUTE PROCEDURE trigf();
```

4.1.4 L'exemple des étudiants

```
CREATE TABLE Etudiants (
  Numero          integer  PRIMARY KEY,
  Nom             char(30) NOT NULL,
  Prenom          char(30) NOT NULL,
  DateDeNaissance date     CHECK (DateDeNaissance > '1900-1-1'));

CREATE TABLE AutresPrenoms (
  Numero          integer  REFERENCES Etudiants(Numero)
                           ON DELETE CASCADE,
  Ordre           integer  CHECK (Ordre > 1),
  Prenom          char(30) NOT NULL,
  PRIMARY KEY (Numero, Ordre),
  UNIQUE (Numero, Prenom));
```

4.1.5 Modification du contenu d'une table

L'insertion, la modification et la suppression des n-uplets se font à l'aide des commandes :

```
INSERT INTO NomDeLaTable VALUES (v1, v2, ..., vn);
UPDATE NomDeLaTable SET (NomColonneI = vi, ... NomColonneJ=vJ) [WHERE condition];
DELETE FROM NomDeLaTable  [WHERE condition];
```

La table peut également être affectée lors de la création de la façon suivante :

```
CREATE TABLE NomDeLaTable (
  NomColonne1 TypeDeDonnée,
  ...
  NomColonneN TypeDeDonnée
) AS ResultatRequeteSelection;

INSERT INTO Etudiants VALUES (45, 'DURAND', 'joe', '1987-06-13');
INSERT INTO Etudiants VALUES (35, 'MARTIN', 'jack', '1986-08-1');
INSERT INTO Etudiants VALUES (70, 'DUPOND', 'averell', '1885-8-2'); -- interdit
INSERT INTO Etudiants VALUES (70, 'DUPOND', 'averell', '1990-8-2');
INSERT INTO Etudiants VALUES (55, 'DURAND', 'joe', '1956-05-18');

INSERT INTO AutresPrenoms VALUES (45, 2, 'leo');
INSERT INTO AutresPrenoms VALUES (45, 3, 'ole');
INSERT INTO AutresPrenoms VALUES (35, 2, 'brice');
INSERT INTO AutresPrenoms VALUES (35, 3, 'jack');
```

4.2 Description de schémas externes

Une vue est une table virtuelle, qui permet une visualisation autre des données. La forme générale est :

```
CREATE VIEW NomDeLaVue (NomColonne1 ... NomColonneN) AS ResultatRequeteSelection;
```

4.2.1 L'exemple des étudiants

```
CREATE VIEW ListeEtudiants (Nom, Prenom, Age) AS
  SELECT Nom, Prenom, EXTRACT(YEAR FROM current_date)-EXTRACT(YEAR FROM DateDeNaissance)
  FROM   Etudiants;
```

Chapitre 5

SQL : un langage algébrique de manipulation de données

SQL (Structured Query Language) est le langage de requêtes le plus utilisé aujourd'hui par les SGBD Relationnels. Dans ce chapitre, nous montrons que SQL permet d'exprimer tous les opérateurs de l'algèbre relationnelle.

Les exemples donnés respectent la syntaxe SQL du SGBD `PostgreSQL 8.0.3` Pour un autre SGBD, la syntaxe peut varier légèrement, et même beaucoup si les opérateurs implémentés diffèrent.

5.1 Forme générale d'une requête SQL

`SELECT` *Liste des noms d'attributs du résultat*
`FROM` *Liste de relations contenant les attributs précédents*
`[WHERE` *un prédicat sur l'union des attributs des relations précédentes* `]` ;

5.2 Les opérateurs unaires

5.2.1 La sélection

Soit R une relation sur les attributs $(A_1, A_2, \ldots, A_n)$. Soit p un prédicat élémentaire sur les attributs $(A_1, A_2, \ldots, A_n)$. Le calcul de $\sigma[p](R)$ s'écrit en SQL :

```
SELECT A1, A2, ..., An
FROM   R
WHERE  p;
```

ou plus simplement

```
SELECT *  /* l'etoile signifie tous les attributs */
FROM   R
WHERE  p;
```

Exemple $\sigma[(\texttt{Nom} = \texttt{Nom2})|(\texttt{Datedenaissance} = \texttt{31mars1985})](\texttt{Etudiants})$ s'écrit

```
SELECT *
FROM   Etudiants
WHERE  (Nom='MARTIN') OR (DateDeNaissance='1987-06-13') ;
```

5.2.2 La projection

Soit R une relation sur les attributs $A = \{A_1, A_2, \ldots, A_n\}$. Soit $S = \{S_1, S_2, \ldots, S_m\} \subset A$ un sous-ensemble des attributs. Le calcul de $\pi[S](R)$ ne s'écrit pas en SQL :

```
SELECT S1, S2, ..., Sn
FROM R;
```

mais

```
SELECT DISTINCT S1, S2, ..., Sn
FROM R;
```

car la norme SQL n'indique pas toujours la suppression des doublons dans les résultats des requêtes.

Exemple $\pi[(\texttt{Nom}, \texttt{Prenom})](\texttt{Etudiants})$ s'écrit

```
SELECT DISTINCT Nom, Prenom
FROM   Etudiants;
```

5.2.3 Le renommage

Le renommage explicite

```
SELECT Colonne1 AS XXX, ..., ColonneN
FROM   R;
```

également possible dans la définition des vues.

```
CREATE VIEW NomDeLaVue (NomColonne1 ... NomColonneN) AS
SELECT Colonne1, ..., ColonneN
FROM   R;
```

Le renommage implicite

La notation pointée

```
SELECT R.NomColonneI
FROM   R;
```

qui est équivalent à :

```
SELECT NomColonneI
FROM   R;
```

Le renommage des relations

```
SELECT NomColonneI
FROM   R RR
WHERE  RR.NomColonneK='Dupont';
```

5.3 Les opérateurs binaires

Cours 5

5.3.1 L'union

Soient R^1, R^2 deux relations sur les attributs $(A_1, A_2, \ldots, A_n)$. Le calcul de $R^1 \cup R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1

UNION

SELECT *
FROM   R2
```

Par défaut, les doublons sont éliminés. Si l'on veut les conserver :

```
SELECT *
FROM   R1

UNION ALL

SELECT *
FROM   R2
```

Exemple Tous les prénoms pour tous les étudiants
$\pi[(\texttt{Numero}, \texttt{Prenom})](\texttt{Etudiants}) \cup \pi[(\texttt{Numero}, \texttt{Prenom})](\texttt{AutresPrenoms})$ s'écrit

```
SELECT Numero, Prenom
FROM   Etudiants
UNION
SELECT Numero, Prenom
FROM   AutresPrenoms;
```

5.3.2 La différence

Soient R^1, R^2 deux relations sur les attributs $(A_1, A_2, \ldots, A_n)$. Le calcul de $R^1 - R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1

EXCEPT /* ou bien MINUS */

SELECT *
FROM   R2
```

Remarque : La différence n'est pas toujours un opérateur implémenté.
Exemple Tous les numéros d'étudiants ayant un seul prénom
$\pi[(\texttt{Numero})](\texttt{Etudiants}) - \pi[(\texttt{Numero})](\texttt{AutresPrenoms})$ s'écrit

```
SELECT Numero
FROM   Etudiants
EXCEPT
SELECT Numero
FROM   AutresPrenoms;
```

5.3.3 Le produit cartésien

Soient R^1, R^2 deux relations sur les attributs $(A^1_1, \ldots, A^1_{n_1})$ et $(A^2_1, \ldots, A^2_{n_1})$, tel que $\{A^1_1, \ldots, A^1_{n_1}\} \cap \{A^2_1, \ldots, A^2_{n_1}\} = \emptyset$ Le calcul de $R^1 \times R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1, R2
```

5.4 Les requêtes imbriquées

La composition des opérateurs de l'algèbre s'écrit aisément.

```
SELECT liste1Attributs
FROM   R1, (
  SELECT liste2Attributs
  FROM RR1, (
    SELECT liste3Attributs
    FROM RRR1, RRR2, ... RRRn
  ) AS RR2, RR3, ... RRn
) AS R2, R3, ... Rn
```

5.5 Les opérateurs déduits

5.5.1 L'intersection

Soient R^1, R^2 deux relations sur les attributs $(A_1, A_2, \ldots, A_n)$. Le calcul de $R^1 \cap R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1

INTERSECT

SELECT *
FROM   R2
```

Remarque : L'intersection n'est pas toujours un opérateur implémenté. Il suffit alors de revenir à l'algèbre relationnelle :$R^1 \cap R^2 = R^1 \; - \; (R^1 \; - \; R^2)$

```
SELECT *
FROM   R1
EXCEPT (
  SELECT *
  FROM   R1
  EXCEPT
  SELECT *
  FROM   R2
);
```

Exemple Les erreurs de saisie sur les prénoms
$\pi[(\texttt{Numero}, \texttt{Prenom})](\texttt{Etudiants}) \cap \pi[(\texttt{Numero}, \texttt{Prenom})](\texttt{AutresPrenoms})$ s'écrit

```
SELECT Numero, Prenom
FROM   Etudiants
INTERSECT
SELECT Numero, Prenom
FROM   AutresPrenoms;
```

5.5.2 La jointure naturelle

Soient R^1, R^2 deux relations sur les attributs $(A_1, \ldots, A_n, B_1, \ldots, B_m)$ et $(A_1, \ldots, A_n, C_1, \ldots, C_p)$. Le calcul de $R^1 \bowtie R^2$ s'écrit en SQL :

```
SELECT *
FROM   R1 NATURAL JOIN R2;
```

Remarque : La jointure naturelle n'est pas toujours un opérateur implémenté. Il suffit alors de revenir à l'algèbre relationnelle :

```
SELECT R1.A1, ..., R1.An, R1.B1, ..., R1.Bm, R2.C1, ...,R2.Cp
FROM   R1, R2
WHERE  R1.A1 = R2.A1
   AND R1.A2 = R2.A2
   AND ...
   AND R1.An = R2.An;
```

Remarque : Les SGBD distinguent quelques fois les `NATURAL JOIN`, les `INNER JOIN`, les `OUTER JOIN` et les `CROSS JOIN`. (cf doc)

Exemple `Etudiants` $\bowtie$ `AutresPrenoms` s'écrit

```
SELECT Etudiants.Numero, Nom, Etudiants.Prenom, Ordre, AutresPrenoms.Prenom
FROM   Etudiants, AutresPrenoms
WHERE  Etudiants.Numero = AutresPrenoms.Numero;

SELECT *
FROM   Etudiants NATURAL JOIN  AutresPrenoms;
```

5.5.3 La thêta-jointure

Soient R^1, R^2 deux relations sur les attributs $(A^1_1, \ldots, A^1_{n_1})$ et $(A^2_1, \ldots, A^2_{n_1})$, tel que $\{A^1_1, \ldots, A^1_{n_1}\} \cap \{A^2_1, \ldots, A^2_{n_1}\} = \emptyset$ Soit P un prédicat élémentaire sur l'union des attributs. Le calcul de $R^1 \theta[P] R^2$ s'écrit en SQL

```
SELECT *
FROM   R1, R2
WHERE  P;
```

C'est la syntaxe de base d'une requête SQL.

Exemple Les étudiants ayant le même premier prénom
`Etudiants` $\theta[Prenom = Prenom]$`Etudiants` s'écrit

```
SELECT *
FROM   Etudiants E1, Etudiants E2
WHERE  (E1.prenom = E2.prenom) AND (E1.Numero < E2.Numero);
```

5.5.4 La division

Soient R^1 une relation sur les attributs $(A_1, \ldots, A_n, B_1, \ldots, B_n)$ et R^2 une relation sur $(B_1, \ldots, B_n)$.

Remarque : La division est un opérateur rarement implémenté. Il faut donc revenir à l'algèbre relationnelle :

Le calcul de $R^1 /\ R^2 = \pi[A_i](R^1) - \pi[A_i]((\pi[A_i](R^1) \times R^2) - R^1)$ s'écrit en SQL

```
SELECT DISTINCT A1, A2, ..., An
FROM   R1
  MINUS
SELECT  A1, A2, ..., An
FROM (
  SELECT *
  FROM   (
    SELECT DISTINCT A1, A2, ..., An,
    FROM   R1
```

```
  ), R2
    MINUS
  SELECT *
  FROM   R1
)
```

Exemple Soient les relations suivantes :

```
CREATE TABLE R1 (
  lettre  char(1),
  chiffre integer,
  indice  integer,
  pays    char(1),
PRIMARY KEY (lettre, chiffre, indice, pays)
);

INSERT INTO R1 VALUES ('A', 7, 33, 'F');
INSERT INTO R1 VALUES ('A', 7, 39, 'I');
INSERT INTO R1 VALUES ('A', 7, 34, 'G');
INSERT INTO R1 VALUES ('F', 6, 39, 'I');
INSERT INTO R1 VALUES ('C', 4, 33, 'F');
INSERT INTO R1 VALUES ('C', 4, 39, 'I');

CREATE TABLE R2 (
  indice  integer,
  pays    char(1),
PRIMARY KEY (indice, pays)
);

INSERT INTO R2 VALUES (33, 'F');
INSERT INTO R2 VALUES (39, 'I');
```

La requête de division s'écrit en SQL :

```
-- les couples (lettre, chiffre) pour lesquels
-- tous les tuples de R2 forment des tuples de R1
SELECT DISTINCT lettre, chiffre
FROM   R1
  EXCEPT
SELECT  lettre, chiffre
FROM (
  SELECT *
  FROM   (
    SELECT DISTINCT lettre, chiffre
    FROM   R1
  ) AS PiR1, R2                      -- Alias obligatoire
    EXCEPT
  SELECT *
  FROM   R1
) AS NonEntierR1;                    -- Alias obligatoire
```

Chapitre 6

Théorie de la normalisation

Cours 6

6.1 Quel schéma conceptuel choisir ?

6.1.1 Un exemple

```
CREATE TABLE R (
  A  char(2),
  B  char(2),
  C  char(2),
PRIMARY KEY (A, B)
);
-- Une contrainte pour verifier ((ai = aj) => (ci = cj))

INSERT INTO R VALUES ('a1', 'b1', 'c2');
INSERT INTO R VALUES ('a1', 'b2', 'c2');
INSERT INTO R VALUES ('a2', 'b1', 'c1');

-- Perte d'information
-- sauf si on introduit une contrainte ((ai,bi) = (aj,bj)) => (ci = cj))
SELECT A, B, C
FROM   (SELECT A, B FROM R) AS R1
  NATURAL JOIN
       (SELECT B, C FROM R) AS R2
EXCEPT
SELECT A, B, C
FROM   R;

-- La contrainte devient une clef primaire
-- La clef primaire devient une contrainte inter-relations
SELECT A, B, C
FROM   (SELECT A, C FROM R) AS R1
  NATURAL JOIN
       (SELECT B, C FROM R) AS R2
EXCEPT
SELECT A, B, C
FROM   R;
```

Les problèmes majeurs :

Redondance La relation R possède de l'information redondante sur les attributs A et C.

Perte d'information Les décompositions peuvent créer des n-uplets lors des jointures.

Perte de contraintes Les décompositions peuvent nécessiter la création de contraintes inter-relations.

6.1.2 Les problèmes potentiels d'un schéma conceptuel

Soit la définition 2.4 déjà donnée : Un schéma relationnel se compose de :
- un ensemble de relations. Pour chacune, obligatoirement :
  - un nom,
  - une liste d'attributs et la liste des domaines associés,
  - un identifiant,

  et éventuellement :
  - une liste d'identifiants externes,
  - une contrainte d'intégrité.
- une éventuelle contrainte d'intégrité inter-relations.

et les opérations classiques d'insertion, de suppression et de modification dans une base de données.

Les problèmes potentiels sont liés au besoin de vérifier qu'après toute opération, les données de la base vérifient les contraintes d'intégrité. Il faut donc vérifier lors d'une :

insertion d'un n-uplet : sa présence dans la relation, l'unicité de son identifiant, et éventuellement la présence des identifiants externes et le respect des contraintes d'intégrité, et enfin le respect de l'éventuelle contrainte d'intégrité inter-relations.

suppression d'un n-uplet : que les n-uplets *dépendants* de son existence sont également supprimés. Ces dépendances si elles existent sont exprimées par les contraintes d'intégrités de la relation, les identifiants externes des autres relations et la contrainte inter-relations.

modification d'un n-uplet : que les n-uplets *dépendants* sont également modifiés. Ces dépendances si elles existent sont exprimées par les contraintes d'intégrités de la relation, les identifiants externes des autres relations et la contrainte inter-relations.

Un bon schéma conceptuel est donc un schéma qui *minimise* le temps nécessaire à ces vérifications. En pratique, il faut chercher à diminuer les contraintes inter-relations qui sont les plus coûteuses, puis les clefs étrangères qui génèrent les modifications et/ou les suppressions, tout en conservant les données et les informations sémantiques.

6.2 Dépendances fonctionnelles et décomposition

Notations : Pour simplifier, on notera X l'ensemble d'attributs $\{X_1, \ldots, X_n\}$. Sauf mention spéciale, on aura toujours $X \cap Y = \emptyset$.

Définition 6.1 (Dépendance fonctionnelle (DF))
Soit $R(X,Y,Z)$ une relation.
$X \longrightarrow Y$ est une dépendance fonctionnelle si et seulement si $(x^1 = x^2) \Rightarrow (y^1 = y^2)$.

Intuitivement, si $X \longrightarrow Y$ est une DF, alors X est un identifiant pour Y.

Définition 6.2 (Décomposition sans perte d'information)
Soit $R(X,Y,Z)$ une relation. La décomposition de R en $R1(X,Y)$ et $R2(X,Z)$ est sans perte d'informations si et seulement si $\pi[X,Y](R) \bowtie \pi[X,Z](R) = R$.

Propriété 6.1 (Théorème de Heath)
Soit $R(X,Y,Z)$ une relation. La décomposition de R en $R1(X,Y)$ et $R2(X,Z)$ est sans perte d'informations si et seulement si il existe une DF $X \longrightarrow Y$.

Pour notre exemple, la bonne décomposition est donc :

```
-- La contrainte ((ai = aj) => (ci = cj)) devient une clef primaire
SELECT A, B, C
```

```
FROM   (SELECT A, C FROM R) AS R1
  NATURAL JOIN
       (SELECT A, B FROM R) AS R2
EXCEPT
SELECT A, B, C
FROM   R;
```

Définition 6.3 (Dépendance fonctionnelle irréductible (DFI))
Soit $R(X, Y, Z)$ *une relation.*
- $X \longrightarrow Y$ *est une dépendance fonctionnelle irréductible à droite si et seulement si* $Y = \{Y_i\}$,
- $X \longrightarrow Y$ *est une dépendance fonctionnelle irréductible à gauche si et seulement* $\forall X_i\ (X - \{X_i\}) \not\longrightarrow Y$.
- $X \longrightarrow Y$ *est une dépendance fonctionnelle irréductible si et seulement si elle est irréductible à droite et à gauche.*

Définition 6.4 (Dépendance fonctionnelle déduite)

la réflexivité : *si* $Y \subset X$ *alors* $X \longrightarrow Y$ *est une DF déduite.*

l'augmentation : *Soit* $X \longrightarrow Y$ *une DF, alors* $\forall Z, (X \cup Z) \longrightarrow (Y \cup Z)$ *est une DF déduite.*

la transitivité : *Soient* $X \longrightarrow Y$ *et* $Y \longrightarrow Z$ *deux DFs, alors* $X \longrightarrow Z$ *est une DF déduite.*

6.3 Graphes des dépendances fonctionnelles d'une relation

Remarque : Un ensemble de DFs $FD = \{X \longrightarrow Y\}$ sur une relation $R(A)$ défini un graphe $FD(R)$ sur l'ensemble des parties de A.

Définition 6.5 (Graphe des DFs d'une relation)
Soit $R(A)$ *une relation et* $FD = \{X \longrightarrow Y\}$ *un ensemble de DFs avec* $X \subset A$ *et* $Y \subset A$.
Le graphe $FD^*(R)$ *des dépendances fonctionnelles de* R *est le graphe obtenu par fermeture (clôture) du graphe* $FD(R)$ *en ajoutant toutes les DFs déduites par réflexivité, augmentation et transitivité.*

Définition 6.6 (Ensemble irréductible de DF)
Soit $R(A)$ *une relation et* $FD = \{X \longrightarrow Y\}$ *un ensemble de DFs avec* $X \subset A$ *et* $Y \subset A$.
FD *est irréductible ssi toutes les Dfs sont irréductibles et* $\forall F \subset FD, F^*(R) \neq FD^*(R)$.

Propriété 6.2 (Graphe miminal des DF)
Soit $R(A)$ *une relation et* $FD = \{X \longrightarrow Y\}$ *un ensemble de DFs avec* $X \subset A$ *et* $Y \subset A$.
Il existe F *irréductible tel que* $F^*(R) = FD^*(R)$.

Algorithme $FD \rightarrow F$ **:** Le calcul d'un ensemble irréductible n'est pas difficile, il suffit de :

1. réflexivité + transitivité donnent pour chaque DF non irréductible à droite un ensemble de DFs irréductibles à droite (singleton) équivalent.
2. supprimer les DFs non irréductibles à droite.
3. Pour chaque DF, tester si la suppression d'un attribut A_i à gauche modifie la clôture. Si non, réduire la DF. Toutes les DFs obtenues sont irréductibles.
4. Pour chaque DF, tester si sa suppression modifie la clôture. Si non, enlever la DF. L'ensemble obtenu est irréductible.

Remarque : Le graphe minimal (l'ensemble des DFs irréductible) n'est pas obligatoirement unique (dépend de l'ordre des choix faits dans l'algorithme précédent). Cette non unicité est un des principaux obstacles à une automatisation totale du processus.

6.3.1 Exemple

Cours 7

Soient une relation $FIRST(Fournisseur, Nom, Statut, Ville, Piece, Quantite)$ et ses dépendances fonctionnelles :
- $\{Fournisseur\} \longrightarrow \{Statut, Ville\}$
- $\{Ville\} \longrightarrow \{Statut\}$
- $\{Fournisseur, Piece\} \longrightarrow \{Quantite\}$
- $\{Fournisseur\} \longrightarrow \{Nom\}$
- $\{Nom\} \longrightarrow \{Fournisseur\}$

La construction d'un ensemble irréductible.

1. remplacement des $\rightarrow$ non irréductibles à droite.
   - $\{Fournisseur\} \longrightarrow \{Statut\}$
   - $\{Fournisseur\} \longrightarrow \{Ville\}$
   - $\{Ville\} \longrightarrow \{Statut\}$
   - $\{Fournisseur, Piece\} \longrightarrow \{Quantite\}$
   - $\{Fournisseur\} \longrightarrow \{Nom\}$
   - $\{Nom\} \longrightarrow \{Fournisseur\}$
2. suppression des $\rightarrow$ non irréductibles à droite. *Rien à faire*
3. suppression des attributs inutiles à gauche. *Rien à faire*
4. suppression des $\rightarrow$ déduites.
   - $\{Fournisseur\} \longrightarrow \{Ville\}$
   - $\{Ville\} \longrightarrow \{Statut\}$
   - $\{Fournisseur, Piece\} \longrightarrow \{Quantite\}$
   - $\{Fournisseur\} \longrightarrow \{Nom\}$
   - $\{Nom\} \longrightarrow \{Fournisseur\}$

6.4 Formes normales

Définition 6.7 (Clef candidate)
Soient $R(A)$ une relation et $FD = \{X \longrightarrow Y\}$ un ensemble de DFs avec $X \subset A$ et $Y \subset A$.
$C \subset A$ est une clef candidate de R ssi $(C \longrightarrow A) \in FD^*$ et est irréductible à gauche.

6.4.1 Première forme normale (1NF)

Problème 1 : Dans une table, quel est la signification de deux n-uplets égaux ?

Définition 6.8 (1NF)
Une table *est en 1NF si et seulement si c'est une relation.*

Remarque : Attention, SQL qui manipule en réalité des multi-ensembles peut donc avoir des tables qui ne sont pas en 1NF.

Propriété 6.3 *$FIRST$ est en 1NF ssi il n'y a pas de doublons.*

6.4.2 Seconde forme normale (2NF)

Problème 2 : Dans $FIRST$, pour chaque pièce d'un fournisseur, la ville est renseignée. Cela engendre de la redondance et des difficultés de mise à jour.

Définition 6.9 (2NF)
Soient $R(A)$ une relation et $FD = \{X \longrightarrow Y\}$ un ensemble de DFs. Notons $C^1, \dots C^p$ les clefs candidates de R. $R(A)$ est en 2NF si et seulement si :
- *elle est en 1NF*

- $\forall Z_j \notin \cup_k C^k, C^i \longrightarrow Z_j$ *est irréductible.*

Informellement, les clefs candidates ne contiennent pas de déterminants.

Propriété 6.4
- *$FIRST$ n'est pas en 2NF. En effet $\{Fournisseur, Piece\}$ est la clef et $\{Fournisseur, Piece\} \longrightarrow \{Ville\}$ n'est pas irréductible.*
- *$SECOND(Fournisseur, Statut, Ville)$ et $FNP(Fournisseur, Nom, Piece, Quantite)$ sont en 2NF.*
- *Le théorème de Health appliqué à $\{Fournisseur\} \longrightarrow \{Statut, Ville\}$ indique que la décomposition de $FIRST$ est sans perte d'information.*

6.4.3 Troisième forme normale (3NF)

Problème 3 : Dans $SECOND$, pour chaque fournisseur d'une ville, le statut est identique. Cela engendre de la redondance et des difficultés de mise à jour.

Définition 6.10 (3NF)
Soient $R(A)$ une relation et $FD = \{X \longrightarrow Y\}$ un ensemble de DFs. Notons $C^1, \ldots C^p$ les clefs candidates de R. $R(A)$ est en 3NF si et seulement si :
- *elle est en 2NF,*
- *si l'on pose $Z = A - \cup C^i$, alors $\forall Z_i \in Z,\ (Z - \{Z_i\}) \not\longrightarrow Z_i$.*

Informellement, les attributs d'aucune clef candidate sont indépendants.

Propriété 6.5
- *$SECOND(Fournisseur, Statut, Ville)$ n'est pas en 3NF. En effet $\{Fournisseur\}$ est la clef et $\{Ville\} \longrightarrow \{Statut\}$*
- *$F(Fournisseur, Ville)$, $V(Ville, Statut)$ et $FNP(Fournisseur, Nom, Piece, Quantite)$ sont en 3NF.*
- *Le théorème de Health appliqué à $\{Fournisseur\} \longrightarrow \{Ville\}$ indique que la décomposition de $SECOND$ est sans perte d'information.*

6.4.4 Forme normale de Boyce-Codd (BCNF)

Problème 3 : Dans FNP, pour chaque pièce, le fournisseur détermine le nom et réciproquement. Cela engendre de la redondance et des difficultés de mise à jour.

Définition 6.11 (BCNF)
Soient $R(A)$ une relation et $FD = \{X \longrightarrow Y\}$ un ensemble de DFs. Notons $C^1, \ldots C^p$ les clefs candidates de R. $R(A)$ est en BCNF si et seulement si :
- *si $X \longrightarrow Y$ est irréductible à gauche (et non réduite), alors $\exists i, C^i \subset X$*

Informellement, les déterminants sont des clefs candidates.

Propriété 6.6
- *$FNP(Fournisseur, Nom, Piece, Quantite)$ n'est pas en BCNF. En effet $\{Nom\}$ n'est pas une clef candidate et $\{Nom\} \longrightarrow \{Fournisseur\}$*
- *$FN(Fournisseur, Nom)$ et $FP(Fournisseur, Piece, Quantite)$ sont en BCNF.*
- *Le théorème de Health appliqué à $\{Fournisseur\} \longrightarrow \{Nom\}$ indique que la décomposition de FNP est sans perte d'information.*

6.4.5 Remarques

- $FIRST = F \bowtie V \bowtie FN \bowtie FP$
- La décomposition obtenue n'est pas idéale. On peut regrouper F et FN en une seule relation.
- Les formes normales forment une hiérarchie. $BCNF \subset 3NF \subset 2NF \subset 1NF$
- Cette hiérarchie continue $5NF \subset 4NF \subset BCNF$
- Cette théorie est basée sur la projection-jointure. Il existe d'autres théorie de la normalisation (une est basée sur la sélection-union).

6.4.6 Perte de dépendances fonctionnelles

Soient la relation $EMP(Eleve, Matiere, Professeur)$ et les dépendances fonctionnelles :
- $\{Eleve, Matiere\} \longrightarrow \{Professeur\}$
- $\{Professeur\} \longrightarrow \{Matiere\}$

qui traduisent le fait qu'un professeur n'enseigne qu'une matière, et qu'un élève n'a qu'un professeur pour une matière donnée.

Propriété 6.7
- *EMP est en 3NF mais n'est pas en BCNF.*
- *$PM(Professeur, Matiere)$ $ST(Eleve, Professeur)$ sont en BCNF, mais la dépendance fonctionnelle $\{Eleve, Matiere\} \longrightarrow \{Professeur\}$ est devenue une contrainte inter-relation.*
- *Aucune des deux décompositions (3NF ou BCNF) n'est satisfaisante.*

6.5 Algorithmes de normalisation

Soient $R(A)$ une relation et $FD = \{X \longrightarrow Y\}$ un ensemble de DFs.

6.5.1 Décomposition en 3NF

L'algorithme de décomposition en 3NF préserve l'information et les dépendances fonctionnelles.

1. Initialiser un ensemble $D = \emptyset$. (D contiendra l'ensemble des relations)
2. Calculer $I = irreductible(FD)$. (I n'est pas unique)
3. Calculer $C^1, \ldots C^p$ les clefs candidates de R.
4. Répéter
   (a) Choisir $f \in I, f = X \longrightarrow Y_f$. (ordre peut influer sur le résultat)
   (b) Calculer $F = \{g \in I, g = X \longrightarrow Y_g\}$.
   (c) Calculer $Z = \cup_g Y_g$.
   (d) Calculer $D = D \cup \{\pi[X, Z](R)\}$.
   (e) Calculer $I = I - (F \cup \{fd \in (X \cup Z)\})$.

   jusqu'à $I = \emptyset$.
5. Calculer $B = A - \cup Att(D_i)$ (l'ensemble des attributs non pris en compte dans D).
6. Si $B \neq \emptyset$ alors $D = D \cup \pi[B](R)$.
7. Si aucune relation $d \in D$ ne contient de clefs candidate C_i, alors choisir une clef C_i et faire $D = D \cup \pi[C_i](R)$. (le choix modifie le résultat)

Cours 8

Exemple :

Soient la relation $FIRST(Fournisseur, Nom, Statut, Ville, Piece, Quantite)$ et ses dépendances fonctionnelles :

$\{Fournisseur\}$	$\longrightarrow$	$\{Statut, Ville\}$
$\{Ville\}$	$\longrightarrow$	$\{Statut\}$
$\{Fournisseur, Piece\}$	$\longrightarrow$	$\{Quantite\}$
$\{Fournisseur\}$	$\longrightarrow$	$\{Nom\}$
$\{Nom\}$	$\longrightarrow$	$\{Fournisseur\}$

1. Il existe quatre ensembles irréductibles

$$I = \begin{array}{|lcl|l}
\hline
\{Ville\} & \longrightarrow & \{Statut\} & \\
\{Fournisseur\} & \longrightarrow & \{Nom\} & \\
\{Nom\} & \longrightarrow & \{Fournisseur\} & \\
\hline
\{Fournisseur\} \textit{ ou } \{Nom\} & \longrightarrow & \{Ville\} & (2) \\
\{Fournisseur, Piece\} \textit{ ou } \{Nom, Piece\} & \longrightarrow & \{Quantite\} & (2) \\
\hline
\end{array}$$

2. Il existe deux clefs candidates $\{Fournisseur, Piece\}$ et $\{Nom, Piece\}$. Cela ne dépend pas des ensembles irréductibles.
3. $\{Fournisseur, Piece\} \longrightarrow \{Quantite\}$ ou $\{Nom, Piece\} \longrightarrow \{Quantite\}$
4. $Z = \{Quantite\}$
5. $D = \{\pi[Fournisseur, Piece, Quantite](FIRST)\}$ ou $D = \{\pi[Nom, Piece, Quantite](FIRST)\}$
6. $I = \begin{array}{|lcl|l}
\hline
\{Ville\} & \longrightarrow & \{Statut\} & \\
\{Fournisseur\} & \longrightarrow & \{Nom\} & \\
\{Nom\} & \longrightarrow & \{Fournisseur\} & \\
\hline
\{Fournisseur\} \textit{ ou } \{Nom\} & \longrightarrow & \{Ville\} & (2) \\
\hline
\end{array}$
7. $\{Ville\} \longrightarrow \{Statut\}$
8. $Z = \{Statut\}$
9. $D = D \cup \{\pi[Ville, Statut](FIRST)\}$
10. $I = \begin{array}{|lcl|l}
\hline
\{Fournisseur\} & \longrightarrow & \{Nom\} & \\
\{Nom\} & \longrightarrow & \{Fournisseur\} & \\
\hline
\{Fournisseur\} \textit{ ou } \{Nom\} & \longrightarrow & \{Ville\} & (2) \\
\hline
\end{array}$
11. Si l'on prend le déterminant qui apparaît deux fois :
    (a) $\{Fournisseur\} \longrightarrow \{Ville\}$ ou $\{Nom\} \longrightarrow \{Ville\}$
    (b) $Z = \{Ville, Nom\}$ ou $Z = \{Ville, Fournisseur\}$
    (c) $D = D \cup \{\pi[Fournisseur, Nom, Ville](FIRST)\}$
    (d) $I = \emptyset$

    Si l'on prend le déterminant qui n'apparaît qu'une seule fois :
    (a) $\{Fournisseur\} \longrightarrow \{Nom\}$ ou $\{Nom\} \longrightarrow \{Fournisseur\}$
    (b) $Z = \{Nom\}$ ou $Z = \{Fournisseur\}$
    (c) $D = D \cup \{\pi[Fournisseur, Nom](FIRST)\}$
    (d) $I = \boxed{\{Nom\} \longrightarrow \{Ville\}}$ ou $I = \boxed{\{Fournisseur\} \longrightarrow \{Ville\}}$
    (e) $\{Nom\} \longrightarrow \{Ville\}$ ou $\{Fournisseur\} \longrightarrow \{Ville\}$
    (f) $D = D \cup \{\pi[Nom, Ville](FIRST)\}$ ou $D = D \cup \{\pi[Fournisseur, Ville](FIRST)\}$
    (g) $I = \emptyset$

L'algorithme retourne donc une décomposition parmi les quatre suivantes :

1. $D = \{\pi[Fournisseur, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom, Ville]\}$
2. $D = \{\pi[Fournisseur, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom], \pi[Fournisseur, Ville]\}$
3. $D = \{\pi[Nom, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom, Ville]\}$
4. $D = \{\pi[Nom, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom], \pi[Nom, Ville]\}$

Problème : Il est possible de construire un outil qui propose toutes les solutions. Mais sur une *vraie* BD, le nombre peut être très grand, et le temps de calcul assez long. À l'inverse, un outil qui renvoie une seule proposition risque de retourner une solution non satisfaisante pour le concepteur.

6.5.2 Décomposition en BCNF

L'algorithme de décomposition en BCNF préserve l'information **mais pas toujours** les dépendances fonctionnelles.

1. Initialiser $D = \{R\}$. (D contiendra l'ensemble des relations)
2. Pour chaque $T \in D$
   (a) Calculer $I^T = irreductible(FD(T))$.
   (b) Calculer $C_T^1, \dots C_T^p$ les clefs candidates de T.
   (c) Trouver $f \in I^T, f = X \longrightarrow Y_f$ et $X \neq C_T^i$. (T n'est pas en BCNF)
   (d) Si trouve alors
      i. Calculer $F = \{g \in I^T, g = X \longrightarrow Y_g\}$.
      ii. Calculer $Z = \cup_g Y_g$.
      iii. Calculer $D = (D - \{T\}) \cup \{\pi[X, Z](T), \pi[A^T - Z](T)\}$.

Exemple :
Soient la relation $FIRST(Fournisseur, Nom, Statut, Ville, Piece, Quantite)$ et ses dépendances fonctionnelles :

$\{Fournisseur\}$	$\longrightarrow$	$\{Statut, Ville\}$
$\{Ville\}$	$\longrightarrow$	$\{Statut\}$
$\{Fournisseur, Piece\}$	$\longrightarrow$	$\{Quantite\}$
$\{Fournisseur\}$	$\longrightarrow$	$\{Nom\}$
$\{Nom\}$	$\longrightarrow$	$\{Fournisseur\}$

1. $D = \{FIRST\}$
2. $T = FIRST$.
3. Il existe quatre ensembles irréductibles

$I =$

$\{Ville\}$	$\longrightarrow$	$\{Statut\}$	
$\{Fournisseur\}$	$\longrightarrow$	$\{Nom\}$	
$\{Nom\}$	$\longrightarrow$	$\{Fournisseur\}$	
$\{Fournisseur\}$ *ou* $\{Nom\}$	$\longrightarrow$	$\{Ville\}$	(2)
$\{Fournisseur, Piece\}$ *ou* $\{Nom, Piece\}$	$\longrightarrow$	$\{Quantite\}$	(2)

4. Il existe deux clefs candidates $\{Fournisseur, Piece\}$ et $\{Nom, Piece\}$.
5. On trouve $\{Ville\} \longrightarrow \{Statut\}$ et $\{Ville\} \neq C_T^i$
6. $D = (D - \{FIRST\}) \cup$
   $\{R_1 = \pi[Ville, Statut](FIRST), R_2 = \pi[Fournisseur, Nom, Ville, Piece, Quantite](FIRST)\}$.
7. R_1 est en BCNF.
8. $T = R_2$.
9. Il existe quatre ensembles irréductibles

$\{Fournisseur\}$	$\longrightarrow$	$\{Nom\}$	
$\{Nom\}$	$\longrightarrow$	$\{Fournisseur\}$	
$\{Fournisseur\}$ *ou* $\{Nom\}$	$\longrightarrow$	$\{Ville\}$	(2)
$\{Fournisseur, Piece\}$ *ou* $\{Nom, Piece\}$	$\longrightarrow$	$\{Quantite\}$	(2)

10. Il existe deux clefs candidates $\{Fournisseur, Piece\}$ et $\{Nom, Piece\}$.
11. Si l'on prend le déterminant qui apparaît deux fois $\{Fournisseur\} \longrightarrow \{Ville\}$ ou $\{Nom\} \longrightarrow \{Ville\}$ on obtient :
    (a) $D = \{R_1\} \cup \{R_3 = \pi[Fournisseur, Nom, Ville](R_2), R_4 = \pi[Fournisseur, Piece, Quantite](R_2)\}$
    ou $D = \{R_1\} \cup \{R_3 = \pi[Nom, Fournisseur, Ville](R_2), R_4 = \pi[Nom, Piece, Quantite](R_2)\}$

(b) R_3 et R_4 sont en BCNF.

et si l'on prend le déterminant qui n'apparaît qu'une fois $\{Fournisseur\} \longrightarrow \{Nom\}$ ou $\{Nom\} \longrightarrow \{Fournisseur\}$ on obtient :

(a) $D = \{R_1\} \cup \{R_3 = \pi[Fournisseur, Nom](R_2), R_4 = \pi[Fournisseur, Ville, Piece, Quantite](R_2)\}$ ou $D = \{R_1\} \cup \{R_3 = \pi[Nom, Fournisseur](R_2), R_4 = \pi[Nom, Ville, Piece, Quantite](R_2)\}$

(b) R_3 est en BCNF.

(c) $T = R_4$.

(d) Dans chaque cas, il existe un ensemble irréductible
$$I^T = \boxed{\begin{array}{lll} \{Fournisseur\}\} & \longrightarrow & \{Ville\} \\ \{Fournisseur, Piece\} & \longrightarrow & \{Quantite\} \end{array}} \quad \text{ou} \quad \boxed{\begin{array}{lll} \{Nom\} & \longrightarrow & \{Ville\} \\ \{Nom, Piece\} & \longrightarrow & \{Quantite\} \end{array}}$$

(e) Dans chaque cas, il existe une clef candidate $\{Fournisseur, Piece\}$ ou $\{Nom, Piece\}$.

(f) On trouve $\{Fournisseur\} \longrightarrow \{Ville\}$ ou $\{Nom\} \longrightarrow \{Ville\}$.

(g) $D = \{R_1, R_3\} \cup \{R_5 = \pi[Fournisseur, Ville](R_4), R_6 = \pi[Fournisseur, Piece, Quantite](R_4)\}$ ou $D = \{R_1, R_3\} \cup \{R_5 = \pi[Nom, Ville](R_4), R_6 = \pi[Nom, Piece, Quantite](R_4)\}$

L'algorithme retourne donc une décomposition parmi les quatre suivantes :

1. $D = \{\pi[Fournisseur, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom, Ville]\}$
2. $D = \{\pi[Fournisseur, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom], \pi[Fournisseur, Ville]\}$
3. $D = \{\pi[Nom, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom, Ville]\}$
4. $D = \{\pi[Nom, Piece, Quantite], \pi[Ville, Statut], \pi[Fournisseur, Nom], \pi[Nom, Ville]\}$

Il existe quelques résultats liant clefs candidates et décomposition.

Propriété 6.8 *Soient $R(A)$ une relation et $FD = \{X \longrightarrow Y\}$ un ensemble de DFs. Notons $C^1, \ldots C^p$ les clefs candidates de R.*

- *Si $\forall(i,j), C^i \cap C^j = \emptyset$ alors les relations obtenues par l'algorithme de décomposition en 3NF sont en BCNF.*

6.6 Intérêt de la normalisation

La théorie de la normalisation a pour principal objectif de formaliser des règles de bon sens que tout bon concepteur applique plus ou moins inconsciemment. L'intérêt est cependant double : d'une part formaliser le bon sens lui donne une garantie ; d'autre part cela permet le développement d'outils.

En pratique, il faut chercher à obtenir une décomposition en BCNF. Si il y a perte de dépendances fonctionnelles, c'est au concepteur de choisir entre cette décomposition avec une contrainte inter-relations et une décomposition en 3NF qui possède quelques problèmes de maintenance.

6.7 Modèle conceptuel Entité-Association et normalisation

La plus part des méthodes (Merise, ...) utilisées en conception de bases de données préfère le modèle conceptuel Entité-Association au modèle conceptuel relationnel.

Schématiquement le modèle Entité-Association comprend :

Des entités qui correspondent aux *objets* de la base. Les caractéristiques des objets sont les attributs.

Des associations qui lient les objets.

Les méthodologies préconisent souvent :

- Pour chaque objet, créer une relation. Si la clef primaire est composé, ajouter un identifiant unique de l'objet qui devient alors la clef primaire.
- Pour chaque association, créer une relation qui contient uniquement les clefs primaires de tous les objets en relation.

Ces méthodologies très simples sont efficaces dans la grande majorité des situations car elles conduisent naturellement à des relations 3NF et même souvent BCNF.

Chapitre 7

Les calculs relationnels

Cours 9

7.1 Introduction

Avec l'algèbre relationnelle, une requête est exprimée comme une séquence d'opérations à réaliser à partir des relations de la bd.

Les langages prédicatifs (ou langage de type calcul) exprime une requête par la simple définition du résultat. Ce sont des langages déclaratifs.

7.2 Le calcul de tuples

La partie déclarative associe les variables aux relations :
- $x \in R$, x désignera les tuples de R.
- $x \in (S \cup T)$, x désignera les tuples de S ou bien de T.

Les formules valides sont contruites à partir des règles :

Ra $true$ et $false$ sont des formules valides.

Rb une condition $x.A\ \theta\ a$ où $x.A\ \theta\ y.B$ avec a une constante, $\theta \in \{=, \neq, <, \leq, >, \geq\}$, est une formule valide.

Rc si f, f_1, f_2 sont des formules valides, alors $\{(f), \neg f, f_1 \vee f_2, f_1 \wedge f_2\}$ sont des formules valides.

Rd si f_x est une formule valide où x est une variable libre, alors $\exists x(f_x)$ et $\forall x(f_x)$ sont des formules valides où x est une variable liée.

Re Il n'y a pas d'autre formule valide.

Les expressions sont de la forme $\{x.A, y.B, \ldots, z.D / f_{(x,y,\ldots,z,t,u,\ldots,w)}\}$ où :
- $\{x, y, \ldots, z\}$ sont des variables déclarées de $R_x, R_y, \ldots, R_z$.
- $A, B, \ldots, D$ sont des attributs des relations correspondantes.
- $f_{\{}x, y, \ldots, z, t, u, \ldots, w\}\}$ est une formule valide dans laquelle les variables $\{z, t, u, \ldots, w\}$ sont liées par des quantificateurs.

La sémantique d'une expression est la projection sur les attributs $A, B, \ldots, D$ du produit cartésien des relations $R_x, R_y, \ldots, R_z$ réduit aux tuples pour lesquels la formule $f_{\{}x, y, \ldots, z, t, u, \ldots, w\}\}$ est vraie.

7.2.1 Exemples

Avec les relations initiales sur les étudiants.

La sélection $\sigma[(\texttt{Nom} = \texttt{Nom2}) | (\texttt{Datedenaissance} = \texttt{31mars1985})](\texttt{Etudiants})$ s'écrit
$\{e.Numero, e.Nom, e.Prenom, e.DateDeNaissance / e.Nom = \texttt{Nom2} \wedge e.DateDeNaissance = \texttt{31mars1985}\}$

La projection $\pi[(\texttt{Nom}, \texttt{Prenom})](\texttt{Etudiants})$ s'écrit
$\{e.Nom, e.Prenom / true\}$

7.2.2 Remarques

En fait, on peut montrer que le calcul de tuples à la même pouvoir expressif que l'algèbre relationnel.
Le langage QUEL (défini en 1975 pour le système INGRES) est basé sur le calcul de tuples.

7.3 Le calcul de domaines

Similaire au calcul de tuples, mais les variables portent sur les valeurs des attributs et non sur les tuples.

La partie déclarative associe les variables aux **attributs** :
- $x \in R.A$, x désignera les valeurs de A.
- $x \in (S.A \cup T.B)$, x désignera les valeurs $S.A$ ou bien de $T.B$.

Les formules valides sont contruites à partir des règles :

Ra *true* et *false* sont des formules valides.

Rb une condition $x \; \theta \; a$ où $x \; \theta \; y$ avec a une constante, $\theta \in \{=, \neq, <, \leq, >, \geq\}$, est une formule valide.

Rc une condition d'appartenance $R(A : v_1, B : v_2, \ldots, D : v_k)$ est une formule valide. Elle est vraie si il existe au moins un tuple ayant la bonne valeur dans R.

Rd si f, f_1, f_2 sont des formules valides, alors $\{(f), \neg f, f_1 \vee f_2, f_1 \wedge f_2\}$ sont des formules valides.

Re si f_x est une formule valide où x est une variable libre, alors $\exists x(f_x)$ et $\forall x(f_x)$ sont des formules valides où x est une variable liée.

Rf Il n'y a pas d'autre formule valide.

Les expressions sont de la forme $\{x, y, \ldots, z / f_{(x,y,\ldots,z,t,u,\ldots,w)}\}$ où :
- $\{x, y, \ldots, z\}$ sont des variables déclarées de $R_x.A_x, R_y.A_y, \ldots, R_z.A_z$.
- $f_\{x, y, \ldots, z, t, u, \ldots, w\}\}$ est une formule valide dans laquelle les variables $\{z, t, u, \ldots, w\}$ sont liées par des quantificateurs.

La sémantique d'une expression est l'ensemble des tuples $x, y, \ldots, z$ du produit cartésien des relations $R_x, R_y, \ldots, R_z$ réduit aux tuples pour lesquels la formule $f_\{x, y, \ldots, z, t, u, \ldots, w\}\}$ est vraie.

7.3.1 Exemples

7.3.2 Remarques

En fait, on peut montrer que le calcul des domaines à la même pouvoir expressif que l'algèbre relationnel.
Le langage QBE (Query By Example défini par IBM) est basé sur le calcul des domaines.

Chapitre 8

SQL : un langage prédicatif

Nous avons déjà vu que SQL permet d'exprimer tous les opérateurs de l'algèbre relationnelle. Dans ce chapitre, nous montrons que SQL permet également l'écriture de requêtes sous forme déclarative.

8.1 Forme générale d'une requête SQL emboitée

```
SELECT liste1Attributs
FROM   R1, R2, ..., Rn
WHERE  conditionDeJointure
  AND  expression;
```

8.2 L'appartenance à un ensemble

Le mot clef `IN` correspond à l'opérateur mathématique $\in$ de la théorie des ensembles.

Remarque : L'ensemble doit contenir des valeurs *atomiques*.

```
-- Calcul des étudiants ayant au moins deux prenoms.
SELECT *
FROM   Etudiants
WHERE  Numero IN
       (SELECT Numero
        FROM   AutresPrenoms);

-- Les étudiants ayant au moins deux prénoms
-- ou un prénom qui soit un autre prénom pour un étudiant quelconque.
SELECT *
FROM   Etudiants
WHERE  Numero IN
       (SELECT Numero
        FROM   AutresPrenoms)
   OR  Prenom IN
       (SELECT Prenom
        FROM   AutresPrenoms);
```

8.3 Les quantifications

Les mots clefs `ANY` et `SOME` correspondent à l'existence dans l'ensemble d'au moins un élément satisfaisant la condition.

Le mot clef ALL correspond au fait que tous les éléments de l'ensemble satisfont la condition.

Remarque : Les ensembles doivent contenir des valeurs *atomiques*, et le mot clef ALL ne correspond pas exactement au ∀ mathématique.

Cours 10

```
-- Calcul des étudiants ayant au moins deux prénoms.
SELECT *
FROM   Etudiants
WHERE  Numero = ANY -- idem = SOME et idem IN dans ce cas
       (SELECT Numero
        FROM   AutresPrenoms);

-- Les étudiants dont le prénom est également un second prénom.
SELECT *
FROM   Etudiants
WHERE  Prenom = SOME
       (SELECT Prenom
        FROM   AutresPrenoms
        WHERE  Ordre = 2);

-- Les étudiants dont le prénom n'est le second prénom d'aucun autre.
SELECT *
FROM   Etudiants
WHERE  Prenom <> ALL
       (SELECT Prenom
        FROM   AutresPrenoms
        WHERE  Ordre = 2);
```

8.4 Les conditions sur des ensembles

8.4.1 Les comparaisons entre ensembles

```
-- Les homonymes
SELECT *
FROM   Etudiants
WHERE  ROW(Nom, Prenom)
     =
       (SELECT Nom, Prenom
        FROM   Etudiants AS TEMP
        WHERE  Etudiants.Numero <> TEMP.Numero
          AND  Etudiants.Nom = TEMP.Nom
          AND  Etudiants.Prenom = TEMP.Prenom);

-- Calcul des étudiants dont les prénoms sont inclus dans les seconds prénoms
/*
 * CONTAINS n'est pas implémenté dans PostGres
SELECT *
FROM   Etudiants
WHERE  ((SELECT Prenom
        FROM   AutresPrenoms
        WHERE  Ordre = 2)
     CONTAINS                         -- ou = ou <>
       (SELECT Prenom
        FROM   AutresPrenoms
```

```
        WHERE  Numero = Etudiants.Numero));
*/
```

Remarque : Les ensembles doivent contenir **un seul** tuple et le premier doit être déclaré comme un tuple.

8.4.2 L'existence d'un élément

Le mot clef `EXISTS` retourne `vrai` si et seulement si l'ensemble est non vide.

```
-- Les étudiants ayant au moins deux prénoms
SELECT *
FROM   Etudiants
WHERE  EXISTS
       (SELECT *
        FROM   AutresPrenoms
        WHERE  Etudiants.Numero = AutresPrenoms.Numero);

-- Les étudiants ayant un seul prénom
SELECT *
FROM   Etudiants
WHERE  NOT EXISTS
       (SELECT *
        FROM   AutresPrenoms
        WHERE  Etudiants.Numero = AutresPrenoms.Numero);
```

8.5 Et la division ?

Nous reprenons l'exemple traité lors de la division SQL algébrique.

```
CREATE TABLE R1 (
  lettre  char(1),
  chiffre integer,
  indice  integer,
  pays    char(1),
PRIMARY KEY (lettre, chiffre, indice, pays)
);

INSERT INTO R1 VALUES ('A', 7, 33, 'F');
INSERT INTO R1 VALUES ('A', 7, 39, 'I');
INSERT INTO R1 VALUES ('A', 7, 34, 'G');
INSERT INTO R1 VALUES ('F', 6, 39, 'I');
INSERT INTO R1 VALUES ('C', 4, 33, 'F');
INSERT INTO R1 VALUES ('C', 4, 39, 'I');

CREATE TABLE R2 (
  indice  integer,
  pays    char(1),
PRIMARY KEY (indice, pays)
);

INSERT INTO R2 VALUES (33, 'F');
INSERT INTO R2 VALUES (39, 'I');
```

Soient $R1(X, Y)$ et $R2(Y)$. Si l'on interprète la division comme $R2 \subseteq \pi[Y](R1)$ alors la requête de division pourrait s'écrire en SQL :

```
-- les couples (lettre, chiffre) pour lesquels
-- tous les tuples de R2 forment des tuples de R1
/*
SELECT DISTINCT lettre, chiffre
FROM   R1
WHERE  ((SELECT indice, pays
         FROM   R1 AS R11
         WHERE  R11.lettre  = R1.lettre
           AND  R11.chiffre = R1.chiffre)
        CONTAINS
        (SELECT *
         FROM   R2));
*/
```

Malheureusement `CONTAINS` est rarement implémenté.

Une autre interprétation de la division est $\{x \ / \ \forall y \in R2, \ \exists(a, b) \in R1 \ \wedge \ (x = a \wedge y = b)\}$, mais le quantificateur $\forall$ n'est jamais implémenté.

Heureusement $\forall x, \ f(x) \equiv \ \nexists x, \ \neg f(x)$.

La division devient $\{x \ / \ \nexists y \in R2, \ \nexists(a, b) \in R1 \ \wedge \ (x = a \wedge y = b)\}$, ce qui donne une autre écriture de la division en SQL.

```
-- les couples (lettre, chiffre) pour lesquels
-- tous les tuples de R2 forment des tuples de R1
SELECT DISTINCT lettre, chiffre
FROM   R1
WHERE  NOT EXISTS
       (SELECT *
         FROM   R2
         WHERE  NOT EXISTS
                (SELECT indice, pays
                 FROM   R1 AS R11
                 WHERE  R11.lettre  = R1.lettre
                 AND    R11.chiffre = R1.chiffre
                 AND    R11.indice  = R2.indice
                 AND    R11.pays    = R2.pays));
```

Chapitre 9

SQL : un langage de présentation des données

Dans ce chapitre, nous mentionnons quelques fonctionnalités SQL qui permettent le formatage des données.

9.1 Les fonctions d'agrégation

SQL offre les fonctions d'agrégation usuelles :

cardinal : le mot clef `COUNT` opère sur un ensemble de tuples.

moyenne : le mot clef `AVERAGE` opère sur un ensemble de valeurs prises par un attribut.

minimum : le mot clef `MIN` opère sur un ensemble de valeurs prises par un attribut.

maximum : le mot clef `MAX` opère sur un ensemble de valeurs prises par un attribut.

total : le mot clef `SUM` opère sur un ensemble de valeurs prises par un attribut.

PostGres offre de nombreuses autres fonctions d'agrégation (cf doc).

9.2 Partitionnement des tuples d'une relation

Le mot clef `GROUP BY` de SQL permet de partitionner l'ensemble des tuples d'une relation.

Exemple :

```
-- Le nombre d'étudiants pour chaque premier prénom
SELECT   Prenom, COUNT(DISTINCT Numero)
FROM     Etudiants
GROUP BY Prenom;
```

Le mot clef `HAVING` permet de sélectionner dans une partition seulement certains groupes. **Attention** la condition porte sur le groupe et non sur les tuples.

```
-- Le nombre d'étudiants pour chaque premier prénom si au moins un est jeune.
SELECT   Prenom, COUNT(DISTINCT Numero)
FROM     Etudiants
GROUP BY Prenom
HAVING   MIN(DateDeNaissance) > 1980-1-1;
```

9.3 L'ordonnancement des tuples

SQL manipule des relations, mais l'utilisateur souhaite souvent mettre en page le résultat d'une requête à des fins de lisibilité.

Le mot clef `ORDER BY` permet de trier les tuples du résultat d'une requête.

```
-- Les étudiants par ordres alphabétiques nom, prénom puis age et numéro.
SELECT   *
FROM     Etudiants
ORDER BY Nom, Prenom, DateDeNaissance DESC, Numero;
```

9.4 Autres fonctionnalités

SQL possède de très nombreuses autres fonctionnalités qui ne seront pas détaillées car elles sortent du cadre de ce cours, mais que vous pouvez (devez) apprendre grâce notamment à l'aide en ligne de PostGres.

Exemple :

```
-- Les étudiants nés dans la nuit du 3 au 13 septembre 1752.
SELECT   *
FROM     Etudiants
WHERE    DateDeNaissance BETWEEN '1752-9-4' AND '1752-9-12';
```

Chapitre 10

Gestion de la correction : les transactions

Cours 11

10.1 Cohérence et correction

Définition 10.1 (État cohérent) *Une base de données est cohérente (ou dans un état cohérent) lorsque toutes les données qu'elle contient sont en accord avec les contraintes d'intégrité du schéma conceptuel.*

Propriété 10.1 *Dans un SGBD relationnel, toute requête conserve la cohérence d'une base de données. Si elle s'exécute avec succès, elle transforme l'état cohérent en un nouvel état cohérent ; si elle échoue (erreur de syntaxe, bug SGBD, arrêt machine ...), elle laisse la base de donnée dans son état cohérent.*

Définition 10.2 (État correct) *Une base de données est dans un état correct lorsque qu'elle est dans un état cohérent et que les valeurs des données reflètent exactement le résultat attendu des modifications effectuées.*

Propriété 10.2 *Dans un SGBD relationnel, les requêtes (même en cas de succès) ne conservent pas la correction d'une base de données.*

10.1.1 Un exemple

```
CREATE TABLE Client (
  nom     char(30),
  compte1 integer,
  compte2  integer,
PRIMARY KEY (nom)
);

INSERT INTO Client VALUES ('Dupont', 1, 2);

CREATE TABLE Compte (
  numero  integer,
  valeur  integer,
PRIMARY KEY (numero)
);

INSERT INTO Compte VALUES (1, 3000);
INSERT INTO Compte VALUES (2, 500);

-- Transfert du compte 1 vers le compte 2 de 1000 euros
UPDATE Compte SET valeur = valeur + 1000 WHERE numero = 2;
UPDATE Compte SET valeur = valeur - 1000 WHERE numero = 1;
```

Entre les deux mises à jour, la base de données est cohérente mais non correcte.

10.2 La transaction : une unité logique de travail

Définition 10.3 (Unité logique de travail) *Une unité logique de travail est une suite ordonnée de requêtes qui conserve la correction des données d'une base. C'est une notion sémantique qui ne peut être que définie par l'utilisateur, et qui ne peut pas être définie comme une contrainte d'intégrité.*

Définition 10.4 (Transaction) *Une transaction est une suite ordonnée de requêtes qui se termine de deux façons possibles :*

Succès *Toutes les requêtes de la liste se sont exécutées avec succès. Le SGDB exécute une instruction* `COMMIT` *qui* valide *la transaction.*

Échec *Au moins une requête de la liste ne s'est pas exécutée avec succès. Le SGDB exécute une instruction* `ROLLBACK` *qui* invalide *la transaction et remet la base dans l'état du début de la transaction.*

```
-- Transfert du compte 1 vers le compte 2 de 1000 euros
START TRANSACTION ; -- ou bien BEGIN en Postgres
UPDATE Compte SET valeur = valeur + 1000 WHERE numero = 2;
UPDATE Compte SET valeur = valeur - 1000 WHERE numero = 1;
END TRANSACTION ; -- ou bien END simplement
```

Propriété 10.3 *Dans un SGBD (relationnel), les transactions conservent la correction d'une base de données.*

10.2.1 Les requêtes sont des transactions

Soit la requête `UPDATE Compte SET valeur = valeur + 1000;` qui modifie tous les tuples de la relation. Si un incident système survient *au milieu* de la requête, la base de donnée sera dans un état incorrect.

Une solution consiste à considérer toutes les requêtes comme des transactions. C'est ce que font par défaut les SGBD (ils sont en mode *autocommit*).

10.3 La gestion des transactions

Avec la définition précédente des transactions deux problèmes majeurs subsistent :

1. Comment mettre en œuvre le `ROLLBACK` ?
2. En cas de panne, comment restaurer la base de données dans un état correct ?

10.3.1 Journal et points de validation

Une solution commune aux deux problèmes est basée sur la gestion d'un journal. Un journal est composé principalement de deux choses :

- une redondance non pas des données mais des opérations effectuées sur les données. Seule les requêtes qui modifient le contenu sont enregistrées et souvent c'est le couple (valeurs avant, valeurs après) qui est mémorisé.
- de points de validation (ou points de synchronisation) qui définissent les états corrects de la base.

et est manipulé à l'aide de deux opérations :

`REDO` qui permet *de simuler* l'effet d'une requête grâce à la redondance du journal.

`UNDO` qui permet *d'annuler* l'effet d'une requête grâce à la redondance du journal.

10.3.2 La mise en œuvre d'une transaction

Par exemple les transactions

```
-- Transfert du compte 1 vers le compte 2 de 1000 euros
START TRANSACTION ; -- ou bien BEGIN en Postgres
UPDATE Compte SET valeur = valeur + 1000 WHERE numero = 2;
UPDATE Compte SET valeur = valeur - 1000 WHERE numero = 1;
END TRANSACTION ; -- ou bien END simplement

-- Transfert du compte 1 vers le compte 2 de 1000 euros
START TRANSACTION ; -- ou bien BEGIN en Postgres
UPDATE Compte SET valeur = valeur + 1000 WHERE numero = 2;
UPDATE Compte SET valeur = valeur - 1000 WHERE XXX = 1;   -- erreur d'attribut
END TRANSACTION ; -- ou bien END simplement
```

en plus de l'exécution en mémoire, écrivent le journal de la façon suivante.

```
START TRANSACTION : ecrire(journal, "START TRANSACTION")
UPDATE            : ecrire(journal, "Compte, (valeur, 2) -> (valeur+1000, 2)")
UPDATE            : ecrire(journal, "Compte, (valeur, 1) -> (valeur-1000, 1)")
END TRANSACTION   : COMMIT : ecrire(journal, "Etat correct")
START TRANSACTION : ecrire(journal, "START TRANSACTION")
UPDATE            : ecrire(journal, "Compte, (valeur, 2) -> (valeur+1000, 2)")
UPDATE            : ROLLBACK :
UNDO              : ecrire(journal, "Compte, (valeur, 2) -> (valeur-1000, 2)")
END TRANSACTION   : COMMIT : ecrire(journal, "Etat correct")
```

Remarque : En fait le processus est un peu plus complexe du fait que les opérations se font en mémoire vive et que les écritures sur disques ne sont faites que lors des points de synchronisations (ou bien lorsque les tampons sont pleins). Cela implique que l'écriture physique du journal soit faite avant l'enregistrement physique des données, et cela simplifie le traitement du `ROLLBACK`.

10.3.3 La reprise après une panne

Soit la situation suivante

```
START TRANSACTION : ecrire(journal, "START TRANSACTION")
UPDATE            : ecrire(journal, "Compte, (valeur, 2) -> (valeur+1000, 2)")
UPDATE            : ecrire(journal, "Compte, (valeur, 1) -> (valeur-1000, 1)")
END TRANSACTION   : COMMIT : ecrire(journal, "Etat correct")
START TRANSACTION : ecrire(journal, "START TRANSACTION")
UPDATE            : ecrire(journal, "Com
```

L'information dans le journal indique que la première transaction s'est déroulée normalement, mais pas la seconde. Elle n'indique pas le lien entre l'état de la mémoire principale et les valeurs enregistrées sur disque.

S'il est clair qu'il ne faut pas *valider* la seconde, faut-il ou bien ne faut-il pas refaire la première à partir de la sauvegarde restaurée ?

Une solution est basée sur les points de contrôles. Un point de contrôle est une marque dans le journal qui liste toutes les transactions en cours, et qui garanti qu'à cet instant, les valeurs en mémoire et sur disque sont les mêmes. Suivant les SGBD, les points de contrôle sont faits à date fixe, ou bien après un nombre déterminés de points de validation. Cours 12

Exemple : Soit un point de contrôle pc à la date tc et une défaillance du système à une date $tf > tc$. Il existe cinq types de transactions :

- T1 Les transactions commencées et finies avant tc.
- T2 Les transactions commencées avant tc et finies avant tf.
- T3 Les transactions commencées avant tc et non terminées.
- T4 Les transactions commencées et finies avant tf.
- T5 Les transactions commencées avant tf et non terminées.

Solution : L'algorithme de reprise après une panne est :

1. Restaurer la base.
2. À partir du dernier point de contrôle *pc*
3. initialiser deux ensembles $Undo = \texttt{transactions(pc)}$ et $Redo = \emptyset$
4. À chaque `START TRANSACTION` T_i, faire $Undo = Undo \cup \{T_i\}$
5. À chaque `COMMIT` T_i, faire $Redo = Redo \cup \{T_i\}$ et $Undo = Undo - \{T_i\}$
6. Parcourir en arrière le journal jusqu'à *pc* en effectuant les `UNDO` nécessaires.
7. Parcourir en avant le journal en effectuant les `REDO` nécessaires.

Sur notre exemple, l'algorithme retourne $Undo = \{T3, T5\}$ et $Redo = \{T2, T4\}$.

Chapitre 11

La gestion des accès concurrents

Les SGBD ont étés créés avec comme objectifs de permettre à des applications multiples d'accéder de façons uniforme à des données qui peuvent être distribuées. Une des conséquences est l'accès concurrent aux données.

11.1 Les problèmes potentiels

11.1.1 Les dépendances non valides dues à un échec

Transaction A	temps	Transaction B
	t1	update $< tuple >$
select $< tuple >$	t2	
COMMIT	t3	
	t4	ROLLBACK

L'état est correct, mais la lecture de l'instant t2 correspond à une valeur qui ne sera jamais présente physiquement dans la base. La transaction A n'est pas reproductible.

Transaction A	temps	Transaction B
	t1	update $< tuple >$
update $< tuple >$	t2	
COMMIT	t3	
	t4	ROLLBACK

L'état est correct, mais l'écriture de l'instant t2 validée à l'instant t3 sera annulée à l'instant t4. L'effet de la transaction A est perdu.

11.1.2 Les états incorrects

Transaction A	temps	Transaction B
select $< tuple >$	t1	
	t2	select $< tuple >$
update $< tuple >$	t3	
COMMIT	t4	
	t5	update $< tuple >$
	t6	COMMIT

À l'instant t4, l'état est correct. À l'instant t6, B pense que l'état est correct, mais la mise à jour à l'instant t5 se fait à partir de la valeur obtenue à l'instant t2 et non à partir de la nouvelle valeur. L'effet de la transaction A est perdu. L'état n'est donc pas *correct*.

11.2 La sérialisation

Définition 11.1 *Une exécution concurrente d'un ensemble de transactions est sérialisable si et seulement si il existe un ordonnancement des transactions qui, pour tout état correct, donne après l'exécution séquentielle le même état correct comme résultat.*

Propriété 11.1 *Une exécution sérialisable transforme un état correct en un état correct.*

Propriété 11.2 *Les trois exemples précédent ne sont pas sérialisables.*

11.3 Une solution : Les verrous

Les problèmes sont d'une part dus à des lectures de valeurs non valides ; d'autre part dus à des conflits d'écritures. Une solution consiste à *verrouiller* les objets (relation, tuples, attribut) pendant leurs utilisations par une transaction de sorte que les autres transactions ne puissent pas y accéder.

Cette solution abouti à deux types de verrous : les verrous exclusifs (X locks) utilisés pour les écritures, et les verrous partagés (S locks) utilisés pour les lectures.

11.3.1 Le verrouillage strict à deux phases

À chaque verrou, on associe une structure à trois champs :
- l'état du verrou (L, S, X),
- D l'ensemble des transactions qui détiennent le verrou,
- F une file de demandes de transactions en attente du verrou.

Tous les accès et ordres sont mis dans la file et le protocole gère les verrous en fonction du contenu de F.

$(L, \emptyset, F)$	$head(F) = lecture(T_i)$	$(S, \{T_i\}, F - \{lecture(T_i)\})$
(S, D, F)	$lecture(T_i) \in F$	$(S, D \cup \{T_i\}, F - \{lecture(T_i)\})$
$(L, \emptyset, F)$	$head(F) = ecriture(T_i)$	$(X, \{T_i\}, F - \{ecriture(T_i)\}$
$(S, \{T_i\}, F)$	$head(F) = ecriture(T_i)$	$(X, \{T_i\}, F - \{ecriture(T_i)\}$
(S, D, F)	$COMMIT(T_i) \in F$	$(S, D - \{T_i\}, F - \{COMMIT(T_i)\})$
(S, D, F)	$ROLLBACK(T_i) \in F$	$(S, D - \{T_i\}, F - \{ROLLBACK(T_i)\})$
$(S, \{T_i\}, F)$	$COMMIT(T_i) \in F$	$(L, \emptyset, F - \{COMMIT(T_i)\})$
$(S, \{T_i\}, F)$	$ROLLBACK(T_i) \in F$	$(L, \emptyset, F - \{ROLLBACK(T_i)\})$
$(X, \{T_i\}, F)$	$COMMIT(T_i) \in F$	$(L, \emptyset, F - \{COMMIT(T_i)\})$
$(X, \{T_i\}, F)$	$ROLLBACK(T_i) \in F$	$(L, \emptyset, F - \{ROLLBACK(T_i)\})$

Propriété 11.3 (Théorème du verrouillage à deux phases) *Avec le verrouillage strict à deux phases, toutes les exécutions concurrentes possibles sont sérialisables.*

Propriété 11.4 (Corollaire) *Avec le verrouillage strict à deux phases, toutes les exécutions concurrentes possibles sont correctes.*

11.3.2 Exemples sur les problèmes

Transaction A	temps	Transaction B	Verrou(*tuple*)
	t1.1	dem($update(tuple)$)	$(L, \emptyset, \{update(B)\})$
	t1.2	$update(tuple)$	$(X, \{B\}, \emptyset)$
dem($select(tuple)$)	t2.1		$(X, \{B\}, \{select(A)\})$
	t4.1	dem($ROLLBACK$)	$(X, \{B\}, \{select(A), ROLLBACK(B)\})$
	t4.2	$ROLLBACK$	$(L, \emptyset, \{select(A)\})$
$select(tuple)$	t2.2		$(S, \{A\}, \emptyset)$
dem($COMMIT$)	t3.1		$(S, \{A\}, \{ROLLBACK(A)\})$
$COMMIT$	t3.2		$(L, \emptyset, \emptyset)$

L'état est correct et les deux transitions se sont exécutées.

Transaction A	temps	Transaction B	Verrou($tuple$)
$select(tuple)$	t1		$(S, \{A\}, \emptyset)$
	t2	$select(tuple)$	$(S, \{A, B\}, \emptyset)$
dem($update(tuple)$)	t3.1		$(S, \{A, B\}, \{update(A)\})$
	t5.1	dem($update(tuple)$)	$(S, \{A, B\}, \{update(A), update(B)\})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Les updates ne pourront jamais être sélectionnés. Les transactions ne peuvent pas terminées, c'est une situation d'interblocage.

11.4 Les blocages

Définition 11.2 (interblocage) *Un ensemble de transactions est interbloqué lorsqu'il existe un cycle dans le graphe d'attente défini par :*

- *les arcs $(T_i, tuple_j)$ qui indique que T_i attend un verrou sur $tuple_j$,*
- *les arcs $(tuple_i, T_j)$ qui indique que $tuple_i$ est détenu par T_j,*

L'exemple le plus classique est :

Transaction A	temps	Transaction B	Verrou($tuple1$)	Verrou($tuple2$)
$update(tuple1)$	t1		$(X, \{A\}, \emptyset)$	$(L, \emptyset, \emptyset)$
	t2	$update(tuple2)$	$(X, \{A\}, \emptyset)$	$(X, \{B\}, \emptyset)$
dem($update(tuple2)$)	t3.1		$(X, \{A\}, \emptyset)$	$(X, \{B\}, \{A\})$
	t5.1	dem($update(tuple1)$)	$(X, \{A\}, \{B\})$	$(X, \{B\}, \{A\})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Définition 11.3 (attente infinie) *Une attente infinie correspond à une exécution au cours de laquelle une transaction reste toujours dans la file d'attente.*

Un exemple d'attente infinie.

Transaction A	temps	Transaction B	Transaction C	Verrou($tuple$)
$select(tuple)$	t1			$(S, \{A\}, \emptyset)$
	t2	$update(tuple)$		$(S, \{A\}, \{B\})$
	t(3+4i)		$select(tuple)$	$(S, \{A, C\}, \{B\})$
$COMMIT$	t(4+4i)			$(S, \{C\}, \{B\})$
$select(tuple)$	t(5+4i)			$(S, \{A, C\}, \{B\})$
	t(6+4i)		$COMMIT$	$(S, \{A\}, \{B\})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Propriété 11.5 *Avec le verrouillage strict à deux phases, les exécutions concurrentes ne sont pas exemptes d'interblocages et d'attentes infinies.*

11.5 Des solutions aux problèmes du blocage

11.5.1 Détection et correction

Pour l'interblocage, il faut mettre en place un algorithme qui calcule le graphe d'attente et détecte la présence d'un cycle. Lorsqu'un cycle est découvert, il faut choisir (arbitrairement) une transaction T_i, l'interrompre et effectuer $ROLLBACK(T_i)$.

Pour l'attente infinie, c'est plus compliqué.

11.5.2 Évitement

Une solution consiste à adapter le protocole à deux phases en mémorisant pour chaque transaction une estampille qui est sa date de création. Cette estampille sert pour soit tuer une transaction, soit s'auto-détruire. Deux versions lorsque (T_i, e_i) demande un verrou sur $tuple_j$ détenu par (T_k, e_k).

Wait-Die : si $e_i < e_k$, T_i attend, sinon T_i meurt.

Wound-Wait : si $e_i < e_k$, T_i blesse T_k, sinon T_i attend.

Dans les deux cas, la transaction tuée redémarre plus tard en gardant son estampille d'origine.

Exemple avec Wait-Die de la séquence conduisant à l'interblocage.

Transaction A	temps	Transaction B	Verrou($tuple1$)	Verrou($tuple2$)
$update(tuple1)$	t1		$(X, \{(A, t1)\}, \emptyset)$	$(L, \emptyset, \emptyset)$
	t2	$update(tuple2)$	$(X, \{(A, t1)\}, \emptyset)$	$(X, \{(B, t2)\}, \emptyset)$
dem($update(tuple2)$)	t3.1		$(X, \{(A, t1)\}, \emptyset)$	$(X, \{(B, t2)\}, \{(A, t1)\})$
	t5.1	dem($update(tuple1)$)	$(X, \{(A, t1)\}, \{(B, t2)\})$	$(X, \{(B, t2)\}, \{(A, t1)\})$
	t6	$ROLLBACK$	$(X, \{(A, t1)\}, \emptyset)$	$(X, \emptyset, \{(A, t1)\})$
$update(tuple2)$	t3.2		$(X, \{(A, t1)\}, \emptyset)$	$(X, \{(A, t1)\}, \emptyset)$
	t7	$update(tuple2)$	$(X, \{(A, t1)\}, \emptyset)$	$(X, \{(A, t1)\}, \{(B, t2)\})$
$COMMIT$	t8		$(L, \emptyset, \emptyset)$	$(L, \emptyset, \{(B, t2)\})$
	t9		$(L, \emptyset, \emptyset)$	$(X, \{(B, t2)\}, \emptyset)$
	t10	$update(tuple1)$	$(X, \{(B, t2)\}, \emptyset)$	$(X, \{(B, t2)\}, \emptyset)$
	t11	$COMMIT$	$(L, \emptyset, \emptyset)$	$(L, \emptyset, \emptyset)$

Propriété 11.6 *Le protocole de verrouillage à deux phases avec évitement garanti que les exécutions concurrentes de transactions sont correctes et sans interblocage. Avec un seul type de verrou, il n'y a pas d'attente infinie. Avec deux types de verrou, l'attente infinie est possible.*

11.6 Et en pratique

11.6.1 Du point de vue des SGBD

- Il y a peu de SGBD qui implémente le protocole de verrouillage à deux phases avec évitement.
- La plus part implémente le protocole de verrouillage à deux phases (strict ou non).
  - Peu implémente la détection de cycle dans le graphe d'attente.
  - La plus part estime qu'une transaction est bloquée (interblocage ou attente infinie) lorsqu'elle est depuis longtemps dans une file d'attente d'un verrou. Le SGBD la tue puis la relance.
- La plus part des SGBD trouve que la pose des verrous ralenti trop l'exploitation de la BD.
  - Ils distinguent différents niveau d'isolation des tuples. C'est une heuristique qui statistiquement fonctionne, mais qui peut-être risquée.

11.6.2 Du point de vue SQL

- SQL n'impose pas la pose de verrou (d'autres techniques existent) mais exige le respect de la correction.
- La norme SQL propose plusieurs niveaux d'isolation qui peuvent être spécifiés en début de transaction : `SERIALIZABLE`, `REPEATABLE READ`, `READ COMMITTED` et `READ UNCOMMITTED`.
- PostGreSQL propose les 4 niveaux mais en implémente que deux : `SERIALIZABLE`, `READ COMMITTED`.

11.6.3 Du point de vue exploitation

Une bonne solution souvent utilisée :
- Une base accessible en lecture uniquement. Les verrous sont inutiles.
- Une base pour les modifications.
- Une recopie de la base modifiée à intervalle de temps régulier.