BASES DE DONNÉES, UNE INTRODUCTION AU MODÈLE RELATIONNEL

VERSION 3.4

JUILLET 1999

D. HERMAN, **P. B**URGEVIN, **P. F**RESNAIS

79

Bases de données, une introduction au modèle relationnel

Daniel HERMAN, Patrice BURGEVIN, Placide FRESNAIS

Université de Rennes 1 - Ifsic

Propos liminaires

Ce polycopié a une longue histoire: la première édition (1986) accompagnait un cours de bases de données, dispensé conjointement aux étudiants de Maîtrise d'Informatique (Université de Rennes 1) et aux élèves de la quatrième année de l'option informatique (Institut National des Sciences Appliquées de Rennes). Il comportait deux parties respectivement consacrées aux aspects externes (langages, conception de schémas relationnels) et aux aspects internes (mise en œuvre) des bases de données. La multiplication, au sein de l'Ifsic, de formations demandant une courte initiation au domaine (Dess Double Compétence, Diplôme d'ingénieur de l'Ifsic, licence d'informatique) nous a conduit à ne retenir, pour la seconde édition (1989) que les aspects externes.

L'évolution technique (multiplication des SGBD relationnels, le caractère désormais incontournable de SQL, l'émergence des SGBD orientés objets…) nous ont conduit à une révision majeure.

Le contenu de cette nouvelle édition est fortement influencé par la durée du cours (une quinzaine d'heures) et par le public visé (il s'agit souvent d'étudiants formés sur un profil d'ingénieur système). Nous avons donc souhaité ne retenir d'un domaine vaste que la culture minimum que doit posséder « l'honnête informaticien ». Nous nous sommes donc focalisés sur les fondements du modèle relationnel en privilégiant les aspects linguistiques et les problèmes posés par la conception de bons schémas relationnels. Ces aspects conceptuels forment le cœur de l'ouvrage (chapitre 2 et 4); nous avons cependant, pour la présente édition, jugé utile d'adjoindre à ces aspects conceptuels un certain nombre d'éléments sur l'état actuel de la technologie.

Bibliographie

Un ouvrage de référence

Un ouvrage de référence nous semble s'imposer:

- ULLMAN J.D., *Principles of Database and Knowledge-Base systems,* Computer Science Press, 1988 (volume 1) et 1989 (volume 2).

  Le volume 1 traite des bases de données classiques et relationnelles ; le volume 2 est orienté vers l'optimisation et la logique.

Bibiliographie sur les SGBD relationnels

Nous donnons une bibliographie générale sur les SGBD relationnels, classée selon l'ordre chronologique :

- DELOBEL C., ADIBA M., *Bases de données et systèmes relationnels,* Dunod Informatique 1982.

  Les fondements théoriques des bases de données relationnelles.

- BOUZEGHOUB M., JOUVE M., PUCHERAL P., *Systèmes de bases de données : des techniques d'implantation à la conception de schémas,* Eyrolles 1990.

  Les chapitres de l'ouvrage sont les suivants: organisation physique, SGBD réseau, SGBD relationnel, exécution des requêtes relationnelles, différentes approches possibles en conception de bases de données.

- DATE C.J with DARWEN H., *A guide to the SQL Standard,* 3rd Edition, Addison-Wesley Publishing Company, MA USA Octobre 1992.

  Un livre sur la norme SQL 92; DATE est l'un des pionniers du domaine.

- MELTON J., SIMON A.R., *Understanding the new SQL: a complete guide,* Morgan Kaufmann Publishers, CA USA 1993.

  La norme SQL 92 est présentée et commentée.

- GARDARIN G., *Maîtriser les Bases de Données - Modèles et langages,* Eyrolles 1993.

  Il s'agit d'une synthèse sur les bases de données, très axée sur SQL et l'algèbre relationnelle.

- MIRANDA S., RUOLS A., *Client-serveur, concepts, moteurs SQL et architectures parallèles,* Eyrolles 1994.

  Ce livre fait le point sur les normes, les standards et les SGBD relationnels.

- GARDARIN G., *Bases de données objet et relationnel,* Eyrolles 1999.

  Ce livre couvre un champ assez large des bases de données — objectifs, architectures, implémenation, accès, conception, exploitation — et présente une détaillée des modèles relationnels et relationnels-objets.

Perspectives: les SGBD objet

Vu le développement actuel des SGBD objet, nous avons souhaité donner quelques titres dans ce domaine :

- DELOBEL C., LECLUSE C., RICHARD P., *Bases de données : des systèmes relationnels aux systèmes à objets*, InterÉditions 1991.

  Une étude approfondie de l'approche objet.

- ADIBA M., COLLET C., *Objets et bases de données - Le SGBD O2*, Hermès 1993.

  Les principales notions qui définissent les SGBD objet sont examinées à travers O_2. Ce livre détaille tout le cycle d'utilisation d'O_2, depuis la conception d'une application jusqu'à l'administration des bases.

- BOUZEGHOUB M., GARDARIN G., VALDURIEZ P., *Les objets*, Eyrolles 1997.

  Un livre récapitulatif sur l'approche objet donnant une large part aux méthodes de conception objet; le point sur la normalisation et sur l'extension objet de la méthode Merise.

- CATTELL R.G.G. BARRY D., *The Object Databases Standard : ODMG-97 release 2.0*, Morgan Kaufmann Publishers, CA USA 1997,

  La norme en matière de SGBD objet: modèle de données ODL, langages de requêtes OQL, connexions C++, SmallTalk et Java.

Remerciements

Nous tenons à remercier Jean LE PALMEC dont les notes de cours ont guidé nos premiers pas dans le domaine.

Les nombreux enseignants de l'Ifsic qui ont été associés à ces enseignements ont évidemment apporté leurs contributions à cet ouvrage. Faute de pouvoir les citer tous, nous les remercions chaleureusement.

Daniel HERMAN, Patrice BURGEVIN, Placide FRESNAIS
Rennes
Mars 1996
Novembre 1998
Juillet 1999

Architecture générale du cours

Le cours comporte 6 chapitres ; la table des matières est à la fin de l'ouvrage.

CHAPITRE 1

Les notions de base

1.1 Introduction

1.1.1 Définition informelle et exemples

Nous cherchons dans ce paragraphe à mettre en avant ce qui distingue une base de données d'un ensemble de fichiers partagé par un groupe d'utilisateurs et exploité par un ensemble de programmes.

Définition 1.1 Base de données
Nous dirons qu'une *base de données* est un ensemble *structuré* de données *persistantes,* représentant une *réalité extérieure* au système, partagé par plusieurs *applications* d'une même *entreprise.*

On remarque que :

1. Les informations contenues dans la base représentent une *réalité extérieure au système* (listes d'adresses, états de compte en banque…) ; ce sont des données du monde réel et non des données informatiques (du type mot mémoire, imprimante…).
2. Les données ont une *structure* qui a été définie une fois pour toutes. La définition de la structure est un acte fondateur lourd de conséquences sur l'exploitation ultérieure.
3. Les utilisateurs (plusieurs *applications*) qui se partagent ces fichiers peuvent avoir des préoccupations différentes.
4. Les applications ne sont pas indépendantes ; elles appartiennent à la même *entreprise* (au sens large ; n'importe quoi d'assez gros : Université, Ministère, banque, entreprise).
5. Les données sont conservées de manière permanente (*persistance*) et elles sont disponibles pour chaque application, sans qu'il y ait besoin de les réintroduire dans le système.

Exemple 1.1 Données concernant les étudiants.

Données

- liste des inscrits
- antécédents des étudiants
- liste des cours
- liste des enseignants
- emplois du temps
- relevés des notes
- règlement des contrôles de connaissances
- …

Applications

- gestion des inscriptions
- octroi des bourses
- planning des salles
- jurys d'examens
- …

Entreprise

- Université

Pour aller plus loin, on peut imaginer que les données sont rangées dans des fichiers. Il faut alors remarquer que la liste des informations explicitement stockées ne sont pas les seules informations accessibles. Ainsi, les informations « Dudule suit le cours de BD licence informatique » et « Duschnock est l'enseignant de BD en licence d'informatique », bien qu'éventuellement enregistrées dans des fichiers différents, contiennent l'information « Duschnock enseigne BD à Dudule ».

Une base de données contient donc des informations représentant des objets du monde extérieur ainsi que des *liens sémantiques* entre ces objets (matérialisés par des traits sur la figure 1.1).

1.1.2 Conception d'une base de données

1.1.2.1 Définitions

Nous nous inspirons ici du modèle entité-association dû à CHEN ; le modèle de données de Merise, dû à TARDIEU, est équivalent fonctionnellement à ce premier modèle.

Les éléments constitutifs d'une base de données, lors de la phase de conception de cette base, correspondent aux trois notions d'entité, d'attribut et d'association.

Définition 1.2 Entité

Une *entité* est objet du monde extérieur que l'on peut distinguer, qui a une existence propre, et qui possède des propriétés ou attributs.

Définition 1.3 Attribut

Un *attribut* est le plus petit élément d'information manipulé par l'entreprise et qui a un sens en lui-même.

Définition 1.4 Association

Une *association* est un lien sémantique entre entités.

Exemple 1.2 Gestion d'une entreprise
Dans cet exemple on considère une entreprise structurée en un certain nombre d'établissements.

- Les établissements emploient des personnes; chaque employé ne travaille que dans un seul établissement, mais peut participer à différents projets de l'entreprise.
- Un projet peut regrouper plusieurs employés, du même établissement ou pas.
- Un projet est dirigé par un et un seul employé, et un employé ne peut diriger qu'un seul projet.
- Si le conjoint d'un employé travaille dans l'entreprise, ce lien est enregistré.
- L'entreprise s'adresse à des fournisseurs qui lui fournissent des produits qu'elle stocke dans des entrepôts.
- Un fournisseur, un établissement, un entrepôt sont localisés dans une ville.
- Un produit peut être composé à partir de plusieurs autres et entrer dans la composition de plusieurs autres : c'est la notion de nomenclature.

Sur cet exemple on distingue facilement des entités: employé, produit, projet, établissement, entrepôt.

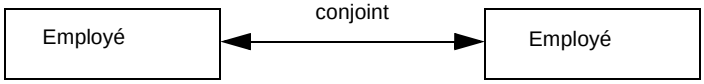
Une entité comme employé possède diverses caractéristiques: matricule, nom, ville de naissance, indice salarial… ce sont les attributs. On ne considère que des attributs mono-valués.

Le fait qu'un employé participe à un projet est un exemple d'association: il y a un lien sémantique entre les entités et les projets.

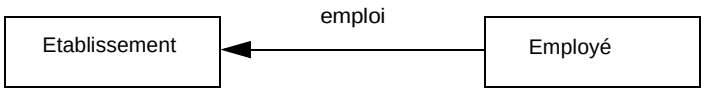
1.1.2.2 Les divers types d'associations

Nous regroupons les associations en quatre catégories et, pour ce faire, nous utilisons une représentation graphique très classique, qui est à rapprocher de celle de dépendance fonctionnelle vue au chapitre 4.

- L'association binaire réciproque ou 1 à 1: le lien qui traduit le fait que deux employés sont mariés est, compte tenu de la législation en vigueur en Europe, un lien de ce type.

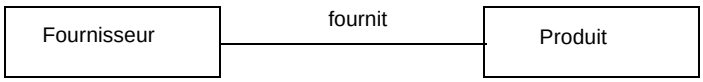


- L'association binaire 1 à plusieurs ou 1 à N: le lien qui relie un établissement aux employés de cet établissement est de cette forme.



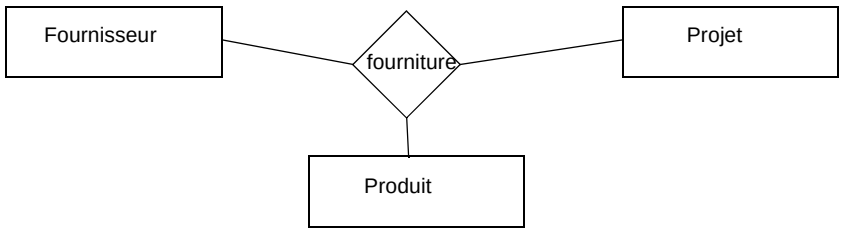
Ce deuxième type d'association est de nature hiérarchique ; nous le retrouverons au chapitre 4, lors de la conception de schémas relationnels, sous l'appellation de dépendance fonctionnelle.

- L'association binaire plusieurs à plusieurs ou N à M : comme le lien de fourniture entre les fournisseurs et les produits, où un fournisseur peut fournir plusieurs produits, et où chaque produit peut être fourni par plusieurs fournisseurs.



Dans notre exemple, il y a de nombreuses associations N à M: le stockage des produits dans les entrepôts, l'utilisation des produits par les projets, l'approvisionnement des projets par les fournisseurs et la fourniture des produits par les fournisseurs.

- L'association n-aire, avec n supérieur à 2. Il existe des associations mettant en jeu plus de deux entités. Ainsi, l'association fourniture représentée ci-après peut contenir une information comme « le fournisseur f2 fournit le produit p3 au projet BD »[1]



1.1.2.3 Instances

Les attributs, les entités et les associations définissent la structure des données, indépendamment des données effectivement stockées dans la base.

L'employé de matricule 1158 constitue une *instance* ou occurrence d'entité ; le matricule est un *attribut clé*, qui permet d'identifier de manière unique chaque occurrence d'entité. Le matricule 1158 constitue une instance d'attribut.

Il peut être rigoureux de parler de type d'entité plutôt que d'entité, de type d'attribut plutôt que d'attribut, auquel cas l'employé de matricule 1158 constitue une entité. Néanmoins, cet emploi du mot type alourdit le vocabulaire et nous ne le retiendrons pas.

Chaque instance d'une association a une existence conditionnée par l'existence des instances des entités reliées (exemple : l'employé de matricule 1158 ne peut participer au projet de code BD que si les deux instances d'employé et de projet existent).

1.1.2.4 Récapitulatif

En utilisant le formalisme qui vient d'être présenté, la représentation de la structure des données du problème se fait ainsi :

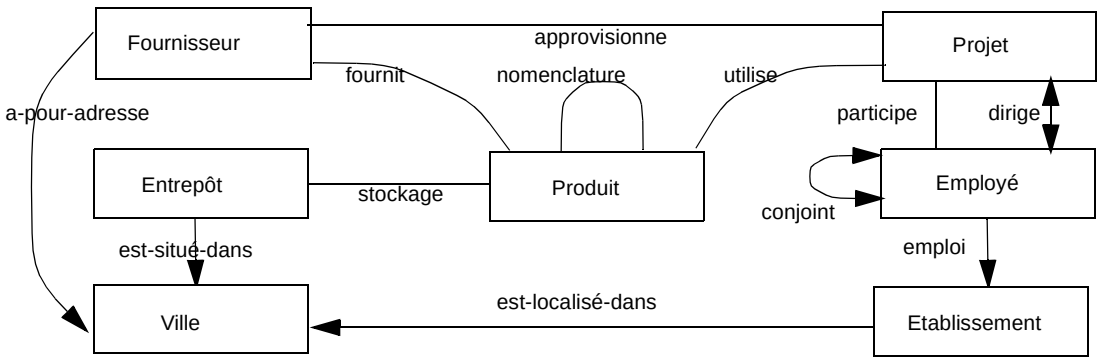


Figure1.1 Exemple de données

1.1.3 Les objectifs d'un SGBD

Parmi tous les ensembles d'informations qui répondent à notre définition d'une base de données, seuls certains sont exploités à l'aide de produits logiciels appelés *Systèmes de Gestion de Bases de Données* (SGBD) ; ce sont ces systèmes qui nous intéressent ici.

1. Cette phrase contient une information plus précise que celle qu'on obtiendrait avec les trois relations binaires: « f2 fournit le produit p3 », « f2 approvisionne le projet BD », « le projet BD utilise le produit p3 ».

L'utilisation d'un SGBD peut permettre d'atteindre plusieurs objectifs :

1. Réduire les redondances d'informations et limiter les saisies.

Exemple

- fichier des étudiants inscrits en sciences, en médecine…
- fichier des étudiants suivant des cours d'informatique.

2. Eliminer des incohérences.

Exemple

- Un étudiant non inscrit suivant un cours d'informatique.

3. Permettre un partage plus souple entre les applications ; répartir une BD sur plusieurs machines ou sites, s'adapter aux besoins nouveaux.

Exemple

- Ouvrir la licence informatique aux médecins

4. Faciliter l'adaptation de standard d'implantation.

Exemple

- Problèmes de fusion des inscrits en sciences et en médecine.

5. Gérer des types de données très variés :
   - numériques, booléens, caractères
   - types textuels avec opérateurs adaptés, bibliographiques avec mots-clé
   - images, sons.
6. Permettre une approche unifiée des problèmes d'intégrité, de confidentialité et de disponibilité.
   - L'intégrité caractérise la fidélité avec laquelle les données représentent la réalité. Assurer l'intégrité consiste à protéger l'utilisateur contre lui-même, contre ses propres erreurs, par la définition d'intervalles de variation, de listes de valeurs, de procédures de contrôle, de contraintes d'intégrité ...
   - Assurer la confidentialité consiste à protéger l'utilisateur contre les autres, par des droits d'accès en lecture ou en écriture, des mots de passe ou du cryptage.
   - La disponibilité d'un système mesure la probabilité qu'il soit en état de fournir son service. Un des aspects permettant d'augmenter la disponibilité est la résistance aux pannes qui consiste à protéger l'utilisateur contre la machine (par la définition de fichiers journaux, de procédures de reprise…).
7. Obtenir des temps de réponse aussi performants que possible. Cet objectif pourra être atteint grâce à :
   - des organisations de données performantes (accès par pointeur, par index, par clé calculée, regroupement de données, ...)
   - une prise en charge du traitement des requêtes par un optimiseur logiciel

- la réalisation de traitements en parallèle, ce qui n'est évidemment pas effectif dans tous les SGBD et ce qui reste un domaine de pointe.

8. Arbitrer plus facilement les conflits d'accès entre consultations et modifications des données.
9. Mais surtout, un objectif fondamental est de permettre une description logique unique, indépendante des caractéristiques d'implantation. Les américains désignent ce phénomène par l'appellation *data independance.*

En effet, les choix d'implantation dûs à l'utilisation de fichiers normaux, présentent de nombreux inconvénients qui se manifestent lors des évolutions que le système doit subir par rapport à sa version initiale.

Exemples

- Un fichier séquentiel trié sur le nom se prête mal à une recherche à partir de l'adresse ou de l'âge.
- Les programmes d'exploitation sont dépendants des choix d'organisation ; toute modification impose un effort énorme de reprogrammation.
- Il est difficile de faire coexister des implantations hétérogènes.

L'indépendance des données permet de donner des langages d'exploitation de la base indépendants des choix d'implantation, donc de permettre une évolution moins coûteuse de la base.

Remarque

Il reste évidemment des problèmes d'évolution et de mauvais choix de départ qui peuvent encore être lourds de conséquences. Disons seulement que plus le SGBD est sophistiqué, plus les obstacles à l'évolution sont de nature logique et non physique.

1.1.4 Architecture d'un SGBD

Le but de ce paragraphe est d'introduire un certain nombre de termes et de sigles d'usage courant.

L'architecture d'un SGBD est décrite par la figure 1.2.

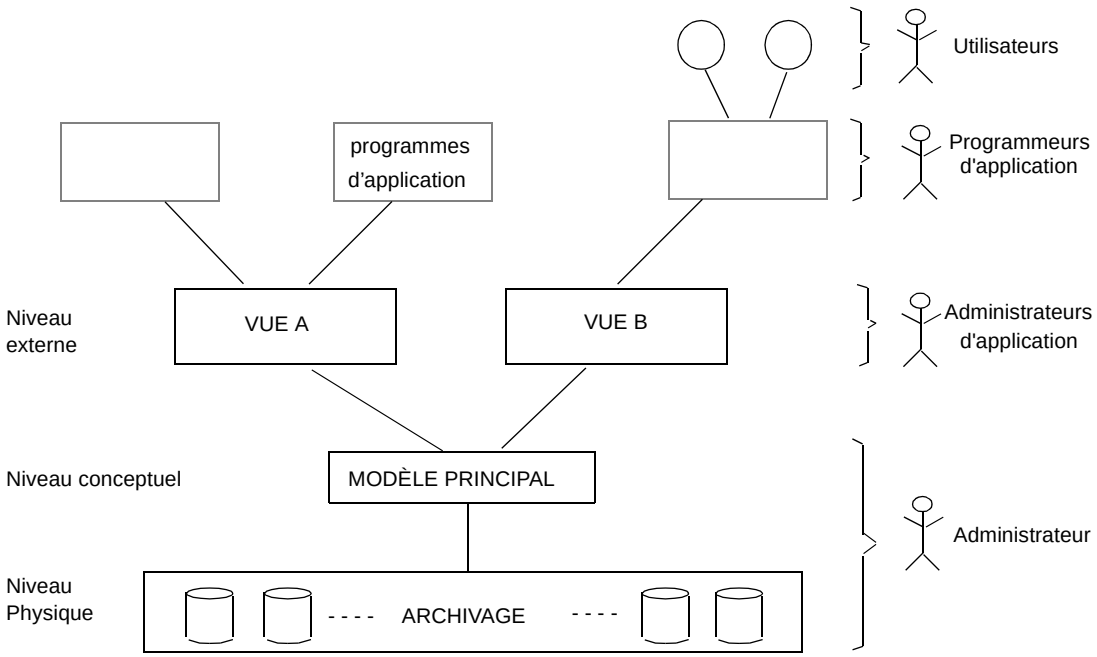


Figure1.2 Architecture d'un SGBD

1.1.4.1 Fonctions principales

1. Description de la base
La description de la base, modèle principal et vues (ou sous-modèles), s'effectue à l'aide d'un langage de définition de données (DDL).

2. Manipulation de la base
Les manipulations de la base (accès, mises à jour) s'effectuent à l'aide d'un langage de manipulation de données (DML).

3. Utilisation de la base
L'utilisation quotidienne de la base peut s'effectuer à l'aide du DML. En fait, le plus souvent, elle s'effectue à l'aide de sous-programmes catalogués, appelés par le biais de commandes.

1.1.4.2 Les différents types d'utilisateurs

On distingue en général :

1. L'administrateur principal de la base (DBA)
Son rôle est fondamental : c'est lui qui définit le modèle principal. Ses choix conditionnent l'évolution ultérieure de la base.

2. L'administrateur d'application
C'est lui qui décrit, à partir du modèle principal, la vue adaptée à l'application qu'il gère.

3. Le programmeur d'application

Son rôle principal est d'établir des bibliothèques de programme d'utilisation de la base (interrogations courantes, mises à jour interactives…). C'est lui qui utilise le DML. En général, ce langage comprend une partie spécifique à la gestion des données (DSL) intégrée au sein d'un langage de programmation standard. Ainsi, toutes les fonctions du SGBD relationnel disponibles sous Oracle sont accessibles à partir du langage C.

1.1.4.3 Description des informations

1. Niveau conceptuel : vue globale - *modèle principal*
2. Niveau externe : application = *vues*
3. Niveau physique : choix d'implantation.

1.2 La base épicerie

Les langages de description de données permettent de décrire les modèles. On distingue quatre modèles principaux :

- le modèle relationnel
- le modèle hiérarchique
- le modèle réseau
- le modèle objet

Nous allons, dans la suite de ce chapitre, passer en revue ces quatre modèles et, pour que le lecteur puisse faire quelques comparaisons, nous les étudions sur le même exemple, la base épicerie. Cet exemple est abondamment réutilisé au cours des chapitres suivants.

Exemple 1.3 La base épicerie
Une entreprise stocke des produits ayant chacun un nom (nom_p), une couleur et une ville d'origine. Chaque produit est identifié par un numéro de code (p). L'entreprise s'adresse à des fournisseurs ayant chacun un nom, résidant dans une ville et pratiquant une certaine remise. Chaque fournisseur est identifié par un numéro de code (f). La fourniture d'un produit par un fournisseur donné se traduit par l'existence d'une quantité (qte) non nulle en stock.

Sur cet exemple, on identifie facilement deux entités, les fournisseurs et les produits et une association entre fournisseur et produit par le biais de la quantité fournie.

En ce qui concerne l'interrogation de la base, nous nous intéressons au schéma d'exécution des deux questions :

Q1 Trouver les numéros des fournisseurs qui fournissent le produit de code p2

Q2 Trouver les numéros des produits fournis par le fournisseur de code f2.

Pour les mises à jour de l'état de la base, nous nous intéressons aux problèmes posés par les trois actions :

M1 Introduire un nouveau fournisseur f7

M2 Le fournisseur f3 ne fournit plus p2

M3 f1 déménage de Paris vers Nice.

Couramment, on appelle M1 une *insertion*, M2 une *destruction* et M3 une *modification*.

1.3 Un langage de description des données : le modèle relationnel

Intuitivement, nous reviendrons sur ce point au chapitre 2, une *relation n-aire* est un tableau à n colonnes. Chaque ligne du tableau est un *n-uplet* et le nom d'une colonne est ce qu'on appelle un *attribut*.

La base de l'exemple 1.3 peut être représentée par les 3 tableaux Fournisseur, Produit, Mafourniture: le problème comporte à l'évidence deux entités, les fournisseurs (tableau Fournisseur) et les produits (tableau Produit). L'état des stocks est donné par le troisième tableau (Mafourniture) qui relie un numéro de fournisseur et un numéro de produit par le biais d'une quantité. On remarque que c'est l'utilisation de noms de colonnes identiques qui permet de traduire l'association entre les entités Fournisseur et Produit.

Fournisseur

f	nom	remise	ville

Produit

p	nom_p	couleur	origine

Mafourniture

f	p	qte

Le modèle des données étant défini, on peut ranger des informations dans la base en remplissant les tableaux. Un contenu possible pour cette base est donné figure 1.3.

Fournisseur

f	nom	remise	ville
f1	Bornibus	5	Paris
f2	Mercier	7	Paris
f3	Colbert	3	Reims
f4	Bossuet	6	Dijon
f5	Tanguy	10	Riec
f6	Dupont	0	Paris

Mafourniture

f	p	qte
f1	p1	1
f1	p4	1
f1	p6	2
f2	p2	1
f3	p2	6
f4	p6	3
f4	p5	7
f1	p5	8
f4	p4	2
f3	p4	1
f2	p4	1
f5	p3	10

Produit

p	nom_p	couleur	origine
p1	cassis	rouge	Dijon
p2	champagne	blanc	Reims
p3	huitre	vert	Riec
p4	moutarde	jaune	Dijon
p5	salade	vert	Nice
p6	cornichon	vert	Dijon
p7	muscadet	blanc	Nantes

Figure1.3 Modèle relationnel.

Toujours très informellement, on peut assimiler chaque relation à un fichier séquentiel, pour lequel on dispose de l'opération **lire** (le suivant).

Interrogation

R1 : **sur** fin de Mafourniture → arrêt ;
repeter
Maf := **lire** (Mafourniture) ;
si Maf.p="p2" **alors** imprimer (Maf.f) **fsi** ;
fin repeter

R2 : **sur** fin de Mafourniture → arrêt ;
repeter
Maf := **lire** (Mafourniture) ;

```
            si Maf.f="f2" alors imprimer (Maf.p) fsi ;
        fin repeter
```

On remarque que les schémas d'exécution de ces deux questions symétriques sont effectivement symétriques.

Mises à jour

M1 consiste à ajouter une ligne au tableau Fournisseur, M2 à supprimer une ligne au tableau Mafourniture, M3 à changer une information et une seule du tableau Fournisseur.

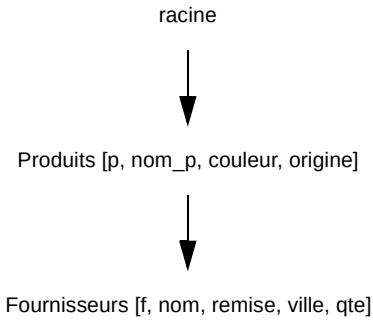
Remarque

Les mises à jour posent cependant des problèmes que nous aborderons au chapitre 4.

1.4 Rappels historiques

1.4.1 Modèle hiérarchique

Le modèle hiérarchique consiste à organiser les diverses entités de la base selon un arbre. Les deux entités évidentes ne laissent que peu de liberté: on peut choisir de relier les fournisseurs aux produits ou les produits aux fournisseurs. Dans une telle optique, il est naturel d'intégrer les quantités fournies aux entités du dernier niveau. On peut donc opter pour l'organisation suivante:



On peut imaginer que les liaisons entre entités sont réalisées par des pointeurs. Selon ce modèle, un contenu de la base est schématisé par la figure 1.4. Nous y remarquons que le fournisseur f6, sans fourniture, ne peut être représenté. Par contre, certains fournisseurs sont représentés plusieurs fois.

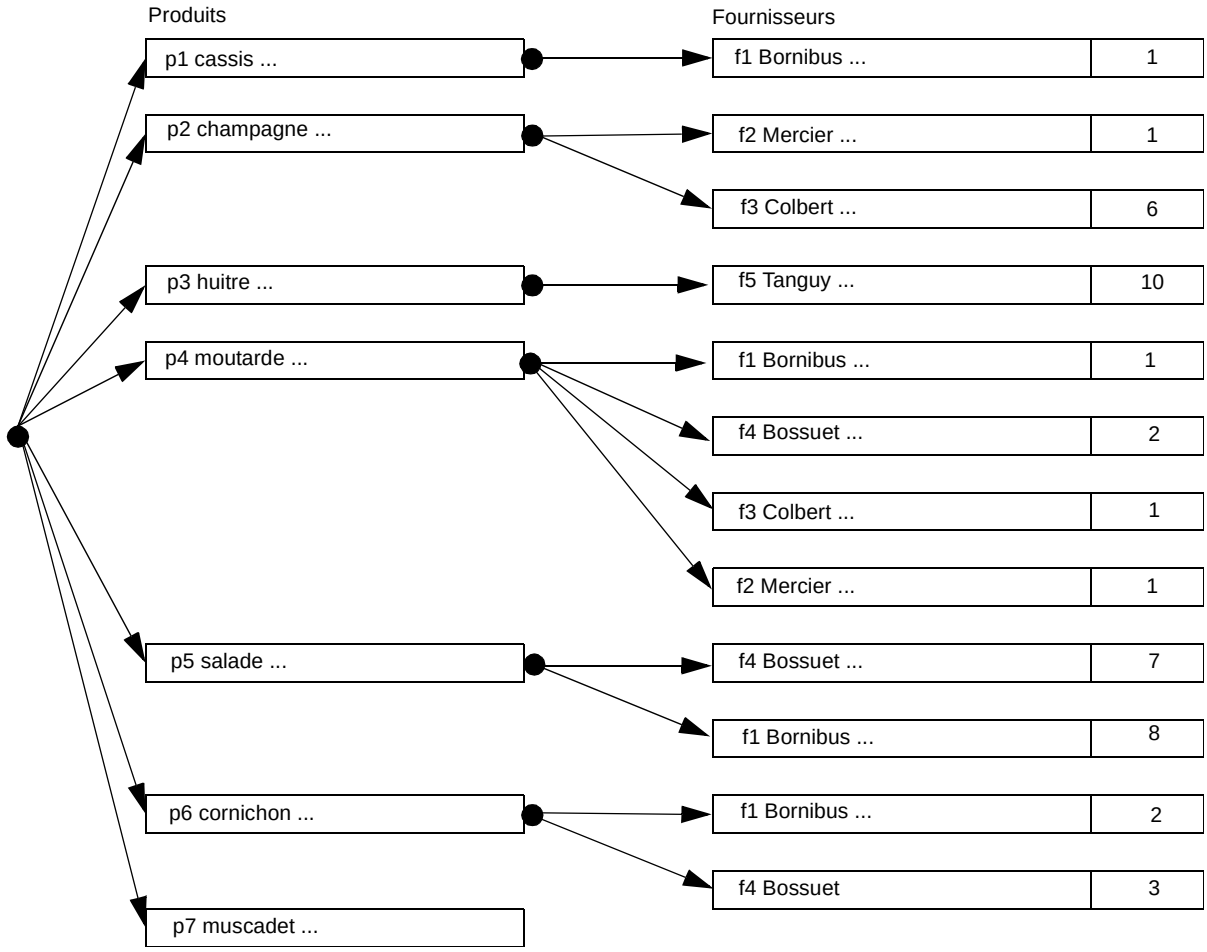


Figure1.4 Modèle hiérarchique.

Interrogation

La base est un arbre et on dispose des primitives fils (le premier) et frère (le suivant). On a alors :

R1 :

```
P := fils (racine)
jqa fini ou trouvé
faire
        quand P=nil sortir par fini;
        quand P.p="p2" sortir par trouvé;
        P:=frère(P);
fait
        trouvé → F:=fils(P);
                tant que F ≠ nil
                faire
                        imprimer(F.f);
                        F:=frère(F)
                fait;
                arrêt;
```

```
            fini → arrêt
      fiter
R2 :
      P:=fils (racine)
      tant que P ≠ nil
      faire
            F:=fils(P);
            jqa trouvé ou fini
            faire
                  quand F=nil sortir par fini;
                  quand F.f="f2" sortir par trouvé;
                  F:=frère(F)
            fait
                  trouvé → imprimer(P.p);
                  fini → rien
            fiter:
            P:=frère(P);
      fait;
      arrêt
```

On remarque que R2 est beaucoup plus coûteux que R1.

Mises à jour

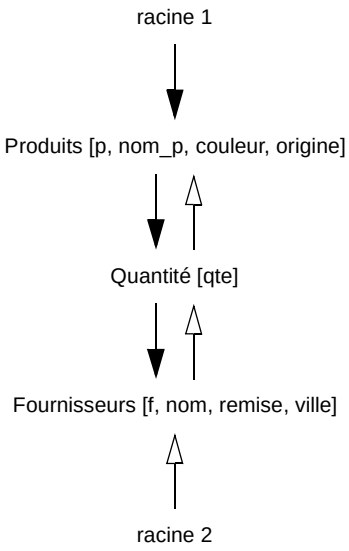
M1 : Comment introduire f7 s'il ne fournit rien ?

M2 : On risque de perdre toute information sur f3.

M3 : On a plusieurs informations à modifier, d'où une recherche dans toute la base.

1.4.2 Modèle Réseau

L'asymétrie inhérente au modèle hiérarchique est la cause des différences entre les algorithmes R1 et R2. Pour pallier cet inconvénient, le modèle réseau permet de relier les diverses entités d'une manière quelconque. On est amené à utiliser une entité quantité reliée à la fois aux produits et aux fournisseurs.



Un contenu partiel selon ce modèle est donné à la figure 1.5. Nous nous limitons ici aux six premières instances du tableau Mafourniture pour avoir un schéma bien lisible. Le lecteur pourra compléter ce schéma à titre d'exercice.

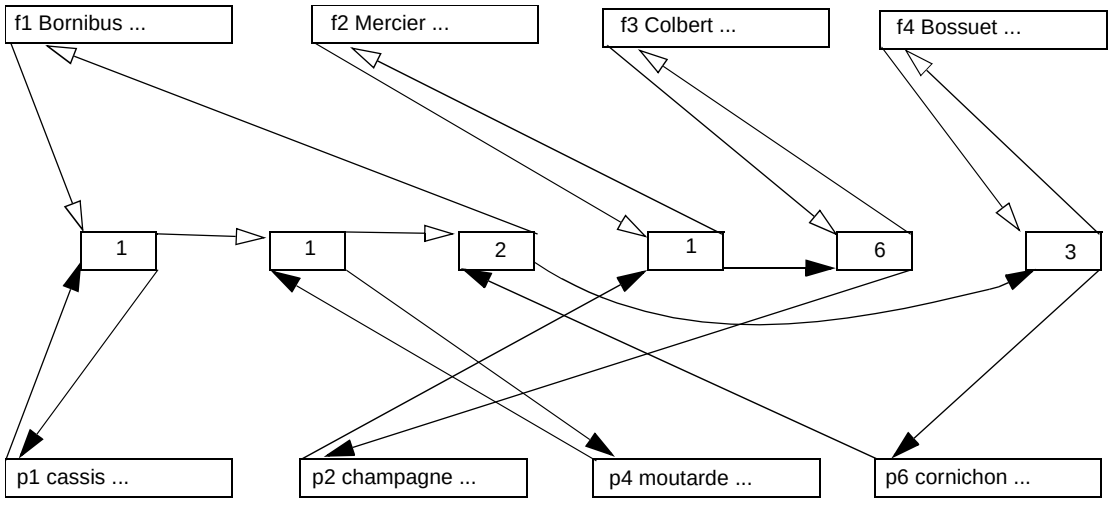


Figure1.5 Modèle Réseau.

On peut considérer qu'on dispose de deux fichiers séquentiels F et P dont les relations sont établies à l'aide de connecteurs (les quantités).

De chaque élément (produit ou fournisseur) est issue une chaîne de connecteurs. On dispose des primitives **tête**, **suivant**, **chaîne-P**, **chaîne-F**, **origine**.

Interrogation
On a :

```
R1 :  P:= lire(P);
      jqa trouvé ou fini
      faire
            quand P=nil sortir par fini;
            quand P.p="p2" sortir par trouvé;
            P:=lire(P);
      fait;
      trouvé → X:=tête(P);
            tant que chaîne-P(X) ≠ P
                  faire
                        F:=origine (chaîne-F (X));
                        imprimer (F.f);
                        X:=suivant (chaîne-P(X));
                  fait;
            arrêt;
      fini → arrêt;
      fiter;
R2 :  F:=lire(F);
      jqa trouvé ou fini
      faire
            quand F=nil sortir par fini;
            quand F.f="f2" sortir par trouvé;
            F:=lire(F);
      fait;
      trouvé → X:=tête(F);
            tant que chaîne-F(X) ≠ F
            faire
                  P:=origine (chaîne-P(X));
                  imprimer (P.p);
                  X:=suivant (chaîne-F(X));
            fait;
            arrêt;
      fini → arrêt;
      fiter;
```

Les deux réponses sont symétriques.

Mises à jour
Les trois mises à jour ne posent pas de problèmes logiques. En revanche, pratiquement, la modification des chaînages n'est pas une opération simple.

1.5 Le modèle objet

1.5.1 Présentation du modèle objet

Le modèle objet est le plus récent et le plus adapté aux structures complexes. Comme ce modèle se développe, nous en résumons les principales caractéristiques.

- Un problème est modélisé en termes de classes, qui regroupent des instances appelées ici exemplaires ou objets. Un type est une notion réduite

de celle de classe, qui permet de structurer l'information. À un type, ne correspondent pas d'objets ayant une existence propre.

- L'encapsulation est un mécanisme destiné à la protection des informations. En une classe, sont réunies les caractéristiques statiques (les attributs) et les caractéristiques dynamiques (les procédures, fonctions ou méthodes) des objets de la classe. Pour chaque composant, que ce soit une donnée ou un traitement, est précisé de manière explicite ce qui est public, accessible de l'extérieur, et ce qui est privé.
- La construction peut se faire par affinage à l'aide de l'héritage ; la surcharge est une redéfinition de l'implantation d'une méthode pour une sous-classe.
- Un objet peut être complexe ; un objet complexe est un objet construit à partir d'objets atomiques par l'application répétée de constructeurs d'objets. Les objets atomiques peuvent être, par exemple, les entiers, les réels, les booléens, les caractères, les chaînes de caractères et les bits. Les constructeurs sont les listes, les ensembles et les n-uplets.
  - Liste = collection ordonnée avec doubles possibles
  - Ensemble = collection non ordonnée
  - N-uplet = regroupement d'attributs qui peut être référencé.
- Tout constructeur peut être appliqué à tout objet : constructeur (constructeur (...)).
- Le système bénéficie de bibliothèques de classes ou de types prédéfinis (DATE, BITMAP, LONG...) et l'utilisateur peut étendre ces classes et types.
- Un objet doit avoir une identité indépendamment de sa valeur.
- Le polymorphisme est la capacité d'appliquer une même fonction (l'addition par exemple) à des objets différents (entiers, réels, nombres complexes). La liaison dynamique est l'association entre un message et sa méthode, en relation avec le polymorphisme, lors de l'exécution et non lors de la compilation.

En résumé, le modèle objet est un modèle sémantique et un modèle d'implantation.

1.5.2 L'exemple épicerie en objet

Modèle des données

Le modèle de données de cet exemple pourrait être ainsi déclaré :

```
Classe Fournisseur  Type n-uplet
      (f : Chaîne,
      nom : Chaîne,
      remise : Réel,
      ville : Chaîne)
Classe Produit Type n-uplet
      (p : Chaîne,
      nom_p : Chaîne,
      couleur : Chaîne,
      origine : Chaîne)
```

Classe Mafourniture Type n-uplet
(four : Fournisseur,
prod : Produit,
qte : Entier)

Chaque objet de Mafourniture fait référence à un objet Fournisseur et à un objet Produit ; un objet est ainsi partagé et non dupliqué. Dans la classe Fournisseur (et / ou dans la classe Produit), une déclaration telle que :

fourniture : Ensemble (Mafourniture)

est possible, sans être habituelle, et peut favoriser par exemple l'expression de contraintes de cardinalité.

Le schéma des objets
La figure 1.6 donne une valeur possible pour les objets de la base épicerie.

Fournisseur

	f	nom	remise	ville
@11	f1	Bornibus	5	Paris
@12	f2	Mercier	7	Paris
@13	f3	Colbert	3	Reims
@14	f4	Bossuet	6	Dijon
@15	f5	Tanguy	10	Riec
@16	f6	Dupont	0	Paris

Mafourniture

four	prod	qte
@11	@51	1
@11	@54	1
@11	@56	2
@12	@52	1
@13	@52	6
@14	@56	3
@14	@55	7
@11	@55	8
@14	@54	2
@13	@54	1
@12	@54	1
@15	@53	10

Produit

	p	nom_p	couleur	origine
@51	p1	cassis	rouge	Dijon
@52	p2	champagne	blanc	Reims
@53	p3	huitre	vert	Riec
@54	p4	moutarde	jaune	Dijon
@55	p5	salade	vert	Nice
@56	p6	cornichon	vert	Dijon
@57	p7	muscadet	blanc	Nantes

Figure 1.6 Objets de la base épicerie.

Interrogation

```
R1 :  sur fin de Mafourniture → arrêt ;
      repeter
            Maf := lire (Mafourniture) ;
            si Maf→prod→p="p2" alors imprimer (Maf→four→f) fsi ;
      fin repeter
R2 :  sur fin de Mafourniture → arrêt ;
      repeter
            Maf := lire (Mafourniture) ;
            si Maf→four→f="f2" alors imprimer (Maf→prod→p) fsi ;
      fin repeter
```

Les schémas d'exécution de ces deux questions sont encore symétriques et proches du relationnel. Cependant, si la question Q1, par exemple, devenait:
Q1' : Trouver les villes des fournisseurs qui fournissent les produits de Paris
son expression serait quasiment inchangée (dans l'instruction conditionnelle de R1, p serait remplacé par origine et f par ville), ce qui ne serait pas le cas en relationnel. Nous laissons de côté ici le problème de l'élimination des doubles qui n'est pas spécifique d'un modèle particulier.

Mises à jour
Les trois mises à jour se réalisent facilement.

1.6 Conclusions, perspectives

Le modèle hiérarchique, directement inspiré par la pratique courante en COBOL, est le plus ancien. Le système IMS (Information Management System) d'IBM est le plus représentatif de cette catégorie, même s'il a bénéficié d'améliorations par rapport au hiérarchique pur.

Le modèle réseau est une proposition émanant du sous groupe DBTG (Data Base Task Group) du CODASYL (groupement autour de COBOL). Les systèmes IDS II (BULL) et IDMS en sont des exemples.

Le modèle relationnel est dû à CODD (1971) et, à la fin des années 70 de nombreuses implantations (ALPHA, SEQUEL, QUERY...) ont été faites dans des laboratoires de recherche. Ces travaux ont eu pour retombées des produits industriels et on peut mesurer le succès du modèle relationnel par le fait que le langage relationnel SQL constitue une norme primordiale. Oracle, Ingres, Sybase et Informix sont représentatifs de ce modèle.

Les travaux sur les SGBD objet se sont développés plus récemment, dans la lignée des langages à objets, pour leur apporter la persistance et une gestion élaborée des données (intégrité, sécurité, ...). Ces SGBD sont particulièrement adaptés à des créneaux spécialisés, tels que CAO, FAO, BD géographiques. Une norme est également définie pour ce modèle, la norme ODMG (Object Database Management Group). Les SGBD O_2 (O_2 Technology) et Versant (Versant Object Technology) sont représentatifs de ce modèle.

Les recherches actuelles portent sur le modèle relationnel qui présente l'avantage de fournir une base mathématique pour définir la sémantique des langages d'interrogation de données, ainsi que sur le modèle objet qui est adapté aux problèmes complexes.

L'avenir lointain est l'utilisation des bases de données en langue naturelle et des bases de données déductives; en attendant, la logique mathématique semble pouvoir donner des perspectives intéressantes.

CHAPITRE 2

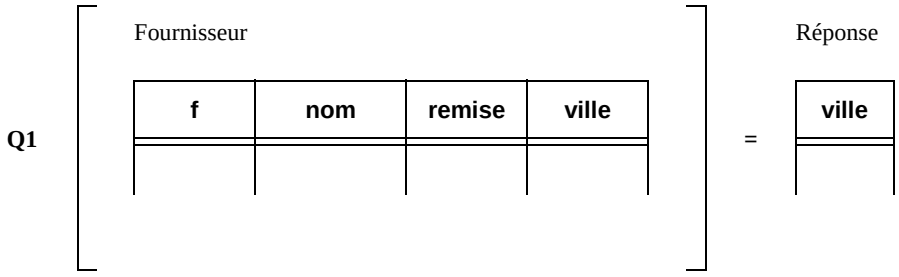
Algèbre relationnelle

2.1 Vers des langages de haut niveau pour manipuler des données

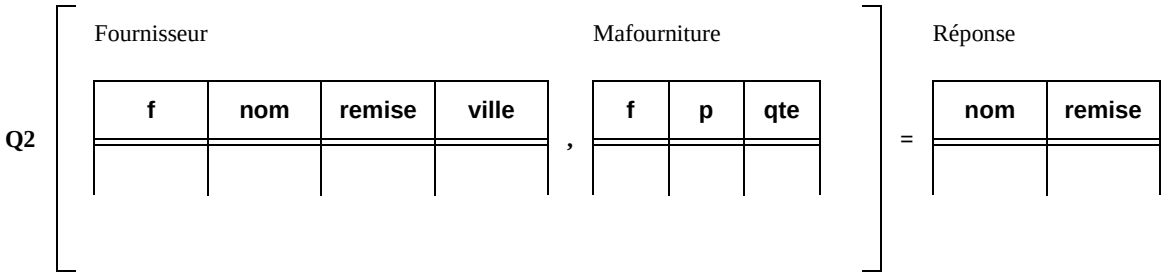
La figure 1.3 du chapitre 1 représente une base de données relationnelle. La base comporte des tableaux dans lesquels sont rangées les données et une interrogation de la base est donc un algorithme dont les paramètres sont des tableaux de la base. On peut imaginer que la réponse à une interrogation est également un tableau.

Exemples

Q1 : Trouver la ville de f1.



Q2 : Trouver les noms et les remises des fournisseurs du produit p2.



Si la réponse à une question quelconque est un tableau, on peut, à son tour, l'utiliser en entrée pour d'autres algorithmes d'interrogation.

Ces constatations mènent à une idée simple: n'est-il pas possible de définir un certain nombre d'opérations élémentaires sur les tableaux de façon à ce qu'une interrogation quelconque puisse s'exprimer en combinant de telles opérations?

C'est l'idée qui a été retenue par CODD et qui conduit à définir l'algèbre relationnelle: une interrogation est une expression formée à l'aide de variables (les tableaux de la base), de constantes et d'opérateurs (les opérateurs de l'algèbre relationnelle).

Au cours de ce chapitre nous précisons la nature de ces objets de base que sont les tableaux, puis nous définissons les opérateurs de l'algèbre relationnelle. Pour conclure, nous évoquons très brièvement la mise en œuvre de requêtes relationnelles.

2.2 Relations

2.2.1 Relation n-aire

Définition 2.1 Soient $D_1,\ldots, D_n$ des ensembles non nécessairement disjoints. Une *relation* R définie sur $D_1,\ldots, D_n$ est un sous-ensemble du produit cartésien $D_1 \times \ldots \times D_n$.

Une relation est donc un ensemble de *n-uplets* $<a_1,\ldots, a_n>$ où, pour chaque i, $a_i \in D_i$.

Comme en informatique on s'intéresse à des relations finies, la liste des n-uplets d'une relation donne bien un *tableau* et nous utilisons indifféremment ces deux noms.

Il est commode (au lieu de les repérer par leur rang comme d'ordinaire en mathématiques) de donner des noms aux colonnes, ce sont les *attributs*.

L'ensemble des valeurs possibles d'un attribut (le D_i correspondant) est appelé *domaine*.

Un descripteur de relation à la forme suivante :

$R(A_1 : d(A_1),..., A_n : d(A_n))$

où

- R est le nom de la relation
- les A_i sont les noms des attributs de la relation
- les $d(A_i)$ sont les domaines associés.

Remarque 1
On peut trouver deux attributs sur le même domaine, par exemple Vol(n : numéro, départ : ville, arrivée : ville).
Dans la suite du texte, quand la mention des domaines n'apporte pas d'information significative, nous les omettons.

Remarque 2
Les valeurs possibles d'un attribut sont supposées être des valeurs atomiques (i.e. non structurées). Des relations respectant cette contrainte sont dites relations en première forme normale.

Nous utiliserons, en général, les premières lettres majuscules de l'alphabet pour désigner des attributs et les dernières pour désigner des ensembles d'attributs. Par abus de notation, nous écrirons souvent XY ou X, Y pour $X \cup Y$ quand X et Y sont disjoints.

Exemple
Avec $R(A_1, A_2, A_3, A_4)$, $X = A_1 \cup A_2$, $Y = A_3 \cup A_4$, nous écrirons R(X, Y) ou R(XY).

Un n-uplet particulier sera noté entre crochets, par exemple $x = <d_1, d_2, d_3>$ ou $x = <A_1 : d_1, A_2 : d_2, A_3 : d_3>$. Sur ce même exemple, $x.A_1$ désignera le 1-uplet $<d_1>$ et si $X = A_1A_3$, x.X désignera le 2-uplet $<d_1, d_3>$. Si $y = <d_1, d_3>$ on pourra écrire $x = <y, d_2>$.

Le contenu des relations d'une base de données informatique varie au cours du temps (adjonctions et suppressions d'informations). Il sera parfois utile de distinguer le descripteur d'une relation qui, lui, est intangible de son contenu courant (ou de ses contenus successifs). Quand cette distinction sera nécessaire, nous noterons en majuscules (R) les descripteurs et en minuscules (r) les contenus ; r est donc un ensemble (de n-uplets).

La valeur d'une relation est la liste des n-uplets qui la composent.

Exemple
La base épicerie.

- Soit les attributs f, nom, remise, ville, p, qte, nom-p, couleur
- Soit les relations :

  Fournisseur (f, nom, remise, ville : cité)
  Produit (p, nom-p, couleur, origine : cité)
  Mafourniture (f, p, qte)

La figure 2.1 donne une valeur possible pour chacune de ces trois relations

Fournisseur

f	nom	remise	ville
f1	Bornibus	5	Paris
f2	Mercier	7	Paris
f3	Colbert	3	Reims
f4	Bossuet	6	Dijon
f5	Tanguy	10	Riec
f6	Dupont	0	Paris

Produit

p	nom_p	couleur	origine
p1	cassis	rouge	Dijon
p2	champagne	blanc	Reims
p3	huitre	vert	Riec
p4	moutarde	jaune	Dijon
p5	salade	vert	Nice
p6	cornichon	vert	Dijon
p7	muscadet	blanc	Nantes

Mafourniture

f	p	qte
f1	p1	1
f1	p4	1
f1	p6	2
f2	p2	1
f3	p2	6
f4	p6	3
f4	p5	7
f1	p5	8
f4	p4	2
f3	p4	1
f2	p4	1
f5	p3	10

Figure2.1 Relations de la base épicerie.

2.2.2 Base de données

Une base de données relationnelle comprend une certain nombre de relations. Cependant, comme toutes les valeurs que peuvent prendre les relations ne représentent pas forcément une réalité du monde extérieur, il est utile d'introduire la notion de contrainte d'intégrité.

Définition 2.2 Pour une relation R, une *contrainte d'intégrité* définit un sous-ensemble des contenus possibles pour R. Ces contenus sont les contenus dits *légaux*.

Exemple

- Toute personne est de sexe masculin ou féminin
- Toute personne a une mère au plus.
- Un même enseignant ne fait pas cours, à la même heure, dans deux salles différentes.

Définition 2.3 Une *base de données* est un triplet *(A, R, I)* où *A* est un ensemble d'attributs, *R* est un ensemble de relations sur ces attributs et *I* un ensemble de contraintes d'intégrité.

2.2.3 Clés

Décider que, pour une relation, un ensemble X d'attributs est une *clé* est un exemple simple de *contrainte d'intégrité.*

Définition 2.4 Une *clé*, pour une relation R, est un ensemble minimum d'attributs identifiant une ligne. Plus précisément :

Soit R(X,Y) , X est une clé pour R si et seulement si

(1) $\forall$ r légal, $\forall$ x,
$(<x, y> \in r \text{ et } <x, y'> \in r) \Rightarrow y = y'$

(2) $\forall A \subset X, \exists$ r légal, $\exists$ a, z, z'
$<a, z> \in r \text{ et } <a, z'> \in r \text{ et } z \neq z'$

Exemples

- {f} est une clé pour Fournisseur;
- {p} est une clé pour Produit;
- {f, p} est une clé pour Mafourniture.

La donnée, pour chaque relation, d'une clé dite *désignée* est donc un exemple de contrainte d'intégrité. Ce sont les seules que nous considérons pour l'instant.

2.3 Algèbre relationnelle

Nous définissons dans ce paragraphe, un certain nombre d'opérateurs relationnels.

Remarque 1 : notations
Les notations utilisées pour représenter les opérateurs sont propres aux auteurs de ce document. Le lecteur pourra trouver dans la littérature d'autres notations.

Remarque 2 : priorité des opérateurs
Quand une expression combine plusieurs opérateurs, à défaut de parenthésage les règles suivantes s'appliquent.

- les opérateurs unaires ont la plus haute priorité. Entre eux, ils sont évalués de gauche à droite.
- les opérateurs binaires sont évalués de gauche à droite.

2.3.1 Premiers opérateurs

2.3.1.1 Projection

Définition 2.5 Soit R(X, Y) la *projection* de R sur Y, notée R[Y] est définie par :

$$<y> \in r[Y] \textbf{ ssi } \exists\, a \text{ tq } <a, y> \in r$$

Interprétation intuitive
On supprime les colonnes non retenues dans la projection et on élimine les doubles.

Exemple
Mafourniture [f]

f
f1
f2
f3
f4
f5

2.3.1.2 Sélection

Définition 2.6 Soit R(X) une relation et *B*(X) un prédicat applicable à tout n-uplet de R, la *sélection* de R par *B,* notée R {*B*} est définie par :

$$<x> \in r\{B\} \textbf{ ssi } <x> \in r \textbf{ et } B(x)$$

Interprétation intuitive
On balaie la relation en ne conservant que les lignes qui vérifient B.

Exemple
Produit {origine = 'Dijon'}

p	nom_p	couleur	origine
p1	cassis	rouge	Dijon
p4	moutarde	jaune	Dijon
p6	cornichon	vert	Dijon

2.3.1.3 Jointure naturelle (composition)

Définition 2.7 Soit S(X, Z) et R(Z, Y), avec $X \cap Y = \emptyset$ et Z éventuellement vide, la *jointure naturelle* de S et de R, notée $S *_{*} R$ est définie par :

$$<x, y, z> \in S *_{*} r \textbf{ ssi } <x, z> \in s \textbf{ et } <z, y> \in r$$

Interprétation intuitive
Pour chaque n-uplet <x, z> de S, on construit dans $S *_{*} R$ autant de k-uplets <x, z, y> qu'il y a de p-uplets <z, y> dans R.

Exemple
Produit $*_{*}$ Mafourniture

nom_p	couleur	origine	p	f	qte
cassis	rouge	Dijon	p1	f1	1
champagne	blanc	Reims	p2	f2	1
champagne	blanc	Reims	p2	f3	6
…	…	…	…	…	…

2.3.2 Opérations ensemblistes: union, intersection, différence

Soit S(X) et R(X). Comme ce sont deux parties du même ensemble, on peut parler de

- $S \cup R$ (union)
- $S \cap R$ (intersection)
- S - R (différence ensembliste)

Exemple
(Mafourniture $*_{*}$ Produit {couleur = 'blanc'})[f] $\cap$ (Mafourniture $*_{*}$ Produit {couleur = 'jaune'})[f]

f
f2
f3

2.3.3 Autres formes de jointures

2.3.3.1 Produit cartésien

Définition 2.8 Soit S(X, Z) et R(Z, Y), avec $X \cap Y = \emptyset$ et Z éventuellement vide. On note S x R le *produit cartésien* de S et de R, défini par

$<x, z_1, z_2, y> \in s \times r$ **ssi** $<x, z_1> \in s$ **et** $<z_2, y> \in r$.

Pour distinguer chacun des attributs dupliqués par cette opération, on utilisera les notations du style R.Z ou S.Z.

Interprétation intuitive
On associe à chaque n-uplet de S chacun des n-uplets de R.

Exemple

M1

f	p
f2	p2
f2	p4

M2

f	p
f4	p4
f4	p5
f4	p6

M1 x M2

M1.f	M1.p	M2.f	M2.p
f2	p2	f4	p4
f2	p2	f4	p5
f2	p2	f4	p6
f2	p4	f4	p4
f2	p4	f4	p5
f2	p4	f4	p6

2.3.3.2 θ–jointure

Définition 2.9 Deux attributs A et B sont dits θ-*compatibles* s'ils appartiennent à des domaines sur lesquels l'opérateur de relation θ est défini.

Définition 2.10 Soit S(X, A) et R(B, Y) avec A et B θ-compatibles, la θ-*jointure* de S et R par rapport à A et B, notée S[[A θ B]]R est une relation sur (X, A, B, Y) définie par :

$<x,a,b,y> \in s\ [[A\ \theta\ B]]r$ ssi $<x,a> \in s$ et $<b,y> \in r$ et $a\ \theta\ b$

Interprétation intuitive
On forme le produit cartésien de S et R ; on élimine tous les éléments ne satisfaisant pas à la relation θ.

Exemple
Fournisseur [[ville = origine]]Produit

f	nom	remise	ville	p	nom_p	couleur	origine
f3	Colbert	3	Reims	p2	champagne	blanc	Reims
f4	Bossuet	6	Dijon	p1	cassis	rouge	Dijon
f4	Bossuet	6	Dijon	p4	moutarde	jaune	Dijon
f4	Bossuet	6	Dijon	p6	cornichon	vert	Dijon
f5	Tanguy	10	Riec	p3	huître	vert	Riec

2.3.3.3 Division

Définition 2.11 Soit R(X,A) et S(B,Y) avec A et B compatibles (sur le même domaine). La *division* de R par S suivant A et B, notée R[[A/B]]S est une relation sur X définie par :

$$\langle x \rangle \in r[[A/B]]s \textbf{ ssi } \forall\, a \in s[B]\ \langle x,a \rangle \in r$$

Interprétation intuitive
On se débarasse de Y dans S, ce qui donne un ensemble E_B de valeurs appartenant à B. On cherche alors dans R les éléments x en relation avec tous les éléments de E_B.

Exemple
P

p	nom_p
p4	moutarde
p5	salade
p6	cornichon

Mafourniture [f, p] [[p/p]] P

f
f1
f4

2.3.4 Exemples d'interrogations

Q1 : Trouver le numéro des fournisseurs qui me fournissent au moins un article

Mafourniture[f]

Q2 : Trouver le numéro des fournisseurs qui ne me fournissent aucun article

Fournisseur[f] - Mafourniture[f]

Q3 : Trouver le numéro des fournisseurs qui me fournissent p6

Mafourniture {p='p6'} [f]

Q4 : Trouver le numéro des fournisseurs qui me fournissent quelque chose d'autre que p6

Mafourniture {p ≠ 'p6'} [f]

Q4' : Trouver le numéro des fournisseurs qui me fournissent quelque chose, mais pas p6

Mafourniture [f] - Mafourniture {p='p6'} [f]

Q5 : Trouver le numéro des fournisseurs qui ne fournissent que p6

Mafourniture [f] - Mafourniture {p ≠ 'p6'} [f]

Q6 : Trouver le numéro des fournisseurs qui me fournissent chacun au moins p4, p5 et p6

Ti(p) = (<p4>, <p5>, <p6>)
Mafourniture [f,p] $[\![p/p]\!] T_i$

ou encore

Mafourniture {p='p4'} [f] ∩ ... ∩ Mafourniture {p='p6'} [f]

Q7 : Trouver le nom et la ville des fournisseurs qui me fournissent tous les produits originaires de Dijon

((Mafourniture [f,p] $[\![p/p]\!]$Tdijon) $_*$ Fournisseur) [nom,ville]

avec

Tdijon = Produit {origine = 'Dijon'} [p]

Q8 : Trouver les numéros des fournisseurs qui me fournissent au moins deux produits.

On renomme Mafourniture par M1 et M2.

(M1 x M2) {M1.f = M2.f ∧ M1.p ≠ M2.p} [M1.f]

2.3.5 Propriétés des opérateurs relationnels

2.3.5.1 Ensemble minimum d'opérateurs

On peut se poser la question de savoir s'il existe un ensemble minimum d'opérateurs relationnels, c'est-à-dire un ensemble tel que

1. on puisse traduire n'importe quelle interrogation (*complétude*);

2. si on enlève un quelconque des opérateurs de l'ensemble, la complétude ne soit plus assurée (*minimalité*).

Nous admettrons que l'ensemble E défini par

- le produit cartésien : x
- la différence ensembliste : -
- l'union ensembliste : $\cup$
- la sélection : { }
- la projection : [ ]

répond à cette question[1].

On peut montrer facilement que les autres opérateurs précédemment introduits s'expriment à l'aide de E.

a) Intersection

$$R \cap S = R - (R - S)$$

b) θ-jointure

$$S[\![A\ \theta\ B]\!]R = (S \times R)\ \{S.A\ \theta\ R.B\}$$

c) jointure naturelle

$$S\ {}_{*}\ R = (S \times R)\ \{S.Z = R.Z\}\ [X, S.Z, Y]$$

Note
Cette définition de la jointure permet de l'étendre au cas où les relations n'ont pas d'attributs communs ; c'est alors le produit cartésien.

d) Division

$$R[\![A/A]\!]S = R[X] - (R[X] \times S[A] - R)\ [X]$$

1. En réalité, l'ensemble E n'est pas complet: on ne peut pas exprimer des interrogations impliquant une fermeture transitive. À titre d'exemple, on pourra essayer de se convaincre qu'il est impossible d'exprimer une relation *Est-un-ancêtre-de* à partir des relations *Est-le-père-de* et *Est-la-mère-de* en se limitant à E. D'ordinaire, dans le cadre des bases de données, on programme les fermetures transitives à l'aide du langage hôte en utilisant plusieurs requêtes relationnelles. Malgré cette limitation, E est suffisant et commode et il sert de référence pour les langages de requêtes.

Exemple

R

X	A
x1	a1
x1	a2
x2	a1
x3	a1
x3	a2
x3	a3
x4	a3

S

A
a1
a2

R[X] x S : « Tout ce qu'il faudrait au moins avoir dans R pour garder tous les x_i ».

R[X] x S

X	A
x1	a1
x1	a2
x2	a1
x2	a2
x3	a1
x3	a2
x4	a1
x4	a2

R[X] x S - R : « Ce qu'il n'y a justement pas dans R (pour garder tous les x_i)».

R[X] x S - R

X	A
x2	a2
x4	a1
x4	a2

(R[X] x S - R) [X] : « Les x_i de R qui ne marchent pas »

R[X] - (R[X] x S - R) [X] : « La réponse ».

L'ensemble E que nous avons défini, présente deux utilités principales :

- Il sert de référence pour tout langage d'interrogation : étant donné un langage I, on s'interroge sur sa complétude en le comparant à l'algèbre relationnelle (avec E). S'il est aussi puissant, on dit que I est complet au sens de CODD.
- On peut démontrer des propriétés sur des expressions relationnelles en n'utilisant que ces 5 opérateurs. En particulier, un procédé de démonstration fréquent consiste à faire une récurrence sur la longueur des expressions : pour passer de n à n+1, on n'a que 5 cas à envisager.

2.3.5.2 Exemples de propriétés

Nous donnons, sans les démontrer, 10 propriétés utiles des opérateurs relationnels. Pour ces propriétés, E, E_1, E_2 et E_3 sont des expressions relationnelles, B, B_1, B_2 des prédicats ne comprenant que des variables ou des constantes, combinées avec $\wedge$, $\vee$, $\neg$ et θ.

P1 : $E_1 \times E_2 = E_2 \times E_1$
P2 : $(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$
P3 : $(E[XY])[X] = E[X]$
P4 : $(E\{B_1\})\{B_2\} = E\{B_1 \wedge B_2\}$
P5 : Quand B porte sur X $(E[X])\{B\} = (E\{B\})[X]$
Quand B porte sur X et Y $(E\{B\})[X] = ((E[XY])\{B\})[X]$
P6 : Si B ne porte que sur les attributs de E_1
$(E_1 \times E_2)\{B\} = E_1\{B\} \times E_2$
Si $B = B_1 \wedge B_2$, B_i portant sur les attributs de E_i
$(E_1 \times E_2)\{B\} = E_1\{B_1\} \times E_2\{B_2\}$
P7 : $(E_1 \cup E_2)\{B\} = E_1\{B\} \cup E_2\{B\}$
P8 : $(E_1 - E_2)\{B\} = E_1\{B\} - E_2\{B\}$
P9 : $(E_1 \times E_2)[Z] = E_1[X] \times E_2[Y]$
où X (resp. Y) représente tous les attributs de Z utilisés dans E_1 (resp. E_2)
P10 : $(E_1 \cup E_2)[X] = E_1[X] \cup E_2[X]$

Ces propriétés peuvent être utilisées dans un cadre général (par exemple montrer l'associativité de $*$) ou pour transformer une interrogation particulière, comme le montre l'exemple suivant, emprunté à la base épicerie.

Exemple
Soit la question : « Trouver les noms des fournisseurs qui me fournissent au moins un produit vert ».

L'expression relationnelle E_1 traduit cette question : E_1 = (Fournisseur $*$ Mafourniture $*$ Produit) {couleur = 'vert'} [nom]

E_1 est très lisible ; on a cependant l'intuition (évidente) qu'on peut se débarrasser des produits non verts avant les jointures. On peut le montrer (par commodité, on renomme Fournisseur par F, Mafourniture par M, Produit par P)

E_1 = (F x M x P) {F.f = M.f} {M.p = P.p} {P.couleur=vert} [nom]
= (((F x M) x P) {P.couleur=vert}) {F.f = M.f} {P.p = M.p}) [nom]
= (((F x M) x (P {couleur=vert})) {F.f = M.f} {P.p = M.P}) [nom]
= (F $*$ M $*$ (P {couleur=vert})) [nom]

Les SGBD utilisent entre autres ce type de propriétés pour optimiser l'évaluation des requêtes relationnelles qui leurs sont confiées.

2.3.6 Mise à jour de la base

On peut tenter d'exprimer les mises à jour de la base à l'aide de l'union et de la différence ensembliste :

insertion
Fournisseur := Fournisseur ∪ <f7, Durand, 5, Lyon>

destruction
Mafourniture := Mafourniture - <f2, p2, 1>

modification
Mafourniture := (Mafourniture - <f2, p2, 1>) ∪ <f2, p2, 4>

Si, dans l'état actuel de l'art, la sémantique des langages d'interrogation de données peut être si précisément exprimée, c'est évidemment parce que la base est considérée comme figée et qu'elle ne comporte pas d'aspects « ennuyeux » comme le dynamisme des variables informatiques. Le cas des mises à jour est « moins clair » : la partie droite des affectations a un sens mathématique qui ne tient pas compte des vérifications des contraintes d'intégrité. Le := reste une opération « mystérieuse ».

2.4 Quelques éléments de mise en œuvre

Il doit être clair que les interprétations intuitives que nous avons données des opérateurs relationnels ne sont pas, en général, des algorithmes efficaces.

Un SGBD dispose de divers moyens d'exécuter raisonnablement les requêtes qui lui sont confiées: il peut utiliser des algorithmes performants de jointure, il peut utiliser des chemins d'accès et il peut transformer les requêtes.

Nous donnons un bref aperçu de chacune de ces trois techniques sur l'exemple de la requête (Fournisseur $_*$ Mafourniture) {p='p6'}.

L'algorithme naïf de jointure consiste en deux boucles imbriquées. Son coût peut s'avérer prohibitif.

2.4.1 Algorithme de jointure par tri et parcours parallèles

Nous donnons un algorithme de jointure qui suppose un tri préalable des deux relations.

Étape 1: tris
- tri de Fournisseur sur f
- tri de Mafourniture sur f

Étape 2: parcours parallèles
- initialiser un parcours de Fournisseur
- initialiser un parcours de Mafourniture
- faire, jusqu'à fin d'une des 2 relations
  - si F.f < M.f alors avancer parcours Fournisseur
  - si F.f > M.f alors avancer parcours Mafourniture
  - en cas d'égalité
    prélecture des n-uplets de Mafourniture qui conviennent
    production de résultats
    avancer parcours Fournisseur

Le coût de cet algorithme de jointure réside essentiellement dans le tri des relations.

2.4.2 Utilisation de chemins d'accès

Index primaire

Le fait de disposer d'une clé désignée par relation permet de les implanter en permettant un accès indexé sur les valeurs de clé. On appelle de tels index, *index primaires.*

Si c'est le cas pour Fournisseur, on dispose d'un autre algorithme, en général moins coûteux que le précédent, pour effectuer la jointure:

- initialiser un parcours de Mafourniture
- faire, jusqu'à fin de Mafourniture
  - accès indexé à Fournisseur pour la clé M.f
  - faire, jusqu'à changement de M.f
    - produire un résultat
    - avancer parcours Mafourniture

Index secondaire

Un *index secondaire* est une structure de donnée permettant de balayer tous les n-uplets d'une relation ayant la même valeur sur un champ donné (l'index secondaire).

Si on dispose pour Mafourniture d'un index secondaire sur le champ p, la requête qui nous intéresse comportant une sélection {p='p6'}, on peut adap-

ter l'algorithme qui précède en balayant l'index secondaire et non la relation.

2.4.3 Transformation de la requête

Dans l'exemple qui précède, pour tirer bénéfice de l'existence d'un index secondaire, nous avons effectué la sélection avant la jointure. Il est clair que le SGBD peut faire de telles transformations; en utilisant des propriétés des opérateurs relationnels, on peut en effet établir que:

(Fournisseur * Mafourniture) {p='p6'} = Fournisseur * (Mafourniture {p='p6'})

CHAPITRE 3

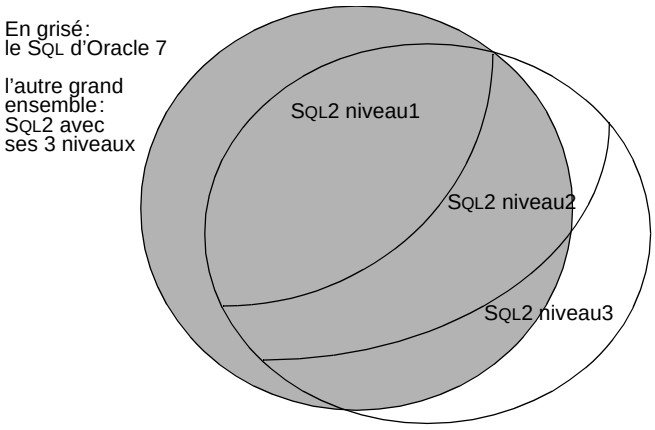
SQL, un langage relationnel

Nous étudions, dans ce chapitre, les principales instructions du langage SQL.

3.1 Introduction

SQL est le langage normalisé dans le domaine des bases de données relationnelles. SQL, conçu au milieu des années 70, vient de *Structured Query Language*, mais cet acronyme n'est plus opératoire; il y a longtemps que le langage permet autre chose que de l'interrogation.

Nous présentons ici la norme SQL2 définie en 1992. Comme ce chapitre est prévu pour permettre l'utilisation de SQL à partir d'Oracle, nous présentons la norme SQL, tout en indiquant quelques différences entre cette norme et le SQL d'Oracle version 7. Il s'agit donc ici pour l'essentiel d'une intersection entre ces deux SQL, sachant que le SQL d'Oracle est compatible avec la norme SQL2 niveau 1, c'est-à dire qu'il l'inclut (or, il y a trois niveaux en SQL2).



Oracle est un SGBD relationnel complet. Il permet de:

1. créer des bases de données,
2. les administrer (entretien, contrôle des accès simultanés...),
3. les consulter et les modifier,
4. développer des applications.

SQL*PLUS, l'interface utilisateur d'Oracle, permet d'utiliser interactivement le langage SQL. Celui-ci offre notamment les fonctions:

- de création de tables,
- d'interrogation,
- de mise à jour.

3.2 Création d'une base de données

Sous SQL, la création d'une base de données consiste en la création des relations de la base. Une relation, appelée table, peut être créée, à l'aide de l'instruction create table.

Les trois instructions suivantes fabriquent les trois relations, vides de données, de la base épicerie:

```
create table  fournisseur      (f varchar(3) primary key,
                                nom varchar(12) not null,
                                remise decimal(2),
                                ville varchar(10) )
create table  produit          (p varchar(3) primary key,
                                nom_p varchar(12) not null,
                                couleur varchar(10),
                                origine varchar(10) )
create table  mafourniture     (f varchar(3) references fournisseur(f),
                                p varchar(3) references produit(p),
                                qte decimal(5) check (qte > 0),
```

constraint fp primary key (f,p))

Remarques

1. Les principaux types de données, communs à SQL2 et à Oracle, sont:

- les nombres exacts: integer ou smallint (nombre entier), decimal(m, d) où m est la taille maximale en chiffres, dont d chiffres après le point décimal;
- les nombres réels: float ou real;
- les chaînes de caractères: char(n) où n est la taille maximale de la chaîne, varchar(n) avec n caractères au plus également et qui permet de n'occuper que la place utile;
- les chaînes de caractères longues: long varchar (jusqu'à 2 giga-octets en Oracle)
- les dates et heures: date (appelé timestamp en SQL2);

2. La spécification not null rend obligatoire la présence d'une valeur pour la colonne concernée, dans tout n-uplet. Inversement, l'absence de cette spécification autorise l'absence de valeur; SQL permet, en effet, le traitement des *valeurs dites nulles.*

3. la Contrainte primary key définit la clé d'une relation. Les valeurs d'une telle clé sont toutes différentes et ne peuvent être des valeurs nulles.

4. Une déclaration de contrainte unique est moins forte que primary key: dans ce cas, l'unicité est assurée, mais avec possibilité de valeur nulle.

5. La contrainte references fournisseur(f) dans la déclaration de la table mafourniture signifie que le champ f de mafourniture est associé à la clé (f) de la relation fournisseur. Ainsi il ne pourra exister un n-uplet $<f_i,...>$ dans mafourniture que si un n-uplet $<f_i,...>$ est présent dans fournisseur. Cela garantit ce qu'on appelle l'intégrité de référence.

6. La contrainte check (qte > 0) signifie que la valeur prise par l' attribut qte est supérieure à 0.

La définition d'une clé a pour effet la création d'un index primaire. Ainsi un index est créé sur le champ f de la relation fournisseur, tout comme sur le champ p de la relation produit. De même, un index est créé sur le couple de champs f et p de la relation mafourniture.

Remarques

- ces index permettent d'améliorer les performances lors des interrogations;
- dans ce but, il est possible de créer des index secondaires sur des champs autres que ceux constituant la clé (instruction create index d'Oracle, présente également dans des extensions de SQL2).

Le remplissage d'une relation peut se faire de deux manières (au moins):

- à l'aide de l'utilitaire de chargement, opérant sur un fichier de commandes et sur un fichier contenant les données initiales,
- à l'aide de l'instruction insert into, permettant d'ajouter un nouvel n-uplet à une relation, par exemple:

```
insert into fournisseur values ('f1', 'Bornibus', 5, 'Paris')
insert into fournisseur values ('f7', 'Martin', NULL, 'Lyon')
```

3.3 Interrogation

La forme générale d'une interrogation est:

```
select a1, ..., ap
 from R1, ..., Rn
 [ where B1 ]
   [ group by b1, ..., bq
   [ having B2 ] ]
[ { union | intersect | minus }  <sélection2> ]
[ order by c1, ..., cr ]
```

Les crochets entourent une clause facultative, et les accolades regroupent plusieurs éléments parmi lesquels un est choisi.
Nous verrons dans ce paragraphe que l'instruction select permet d'une part d'exprimer toutes les opérations de l'algèbre relationnelle, et d'autre part d'autres opérations telles que, par exemple, des calculs portant sur des groupes de n-uplets.
La clause order by permet un tri du résultat.

3.3.1 Construction de base

La construction de base est l'instruction select.

Forme générale (simplifiée)

```
select a1, ..., ap
 from R1, ..., Rn
 where B
```

où les a_i sont des attributs, les R_i des relations et B un prédicat.

Sémantique

$(R_1 \times ... \times R_n) \{ B \} [a_1, ..., a_p]$

En fait la sémantique de l'instruction select diffère de l'expression relationnelle ci-dessus, du point de vue de la projection dans le sens où par défaut il n'y a pas élimination des doubles; si l'on désire éliminer les doubles, il faut faire précéder la liste d'attributs citée dans le select par le mot-clé distinct.

Remarques

- * est une convention qui remplace tous les attributs des relations citées dans le from,
- le where est facultatif.

Exemples

- select * from produit

- select nom
  from fournisseur, mafourniture
  where fournisseur.f = mafourniture.f and mafourniture.p = 'p2'

réalise une jointure naturelle de deux tables, une sélection (p = 'p2') et une projection sur le nom. Une requête équivalente est:

```
select nom
 from fournisseur
  where f in (select f from mafourniture where p = 'p2')
```

En SQL2, il y a d'autres manières d'exprimer la jointure[1].

3.3.2 Opérations ensemblistes

Forme générale

```
<sélection1>    union       <sélection2>
<sélection1>    intersect   <sélection2>
<sélection1>    minus       <sélection2>
```

Exemple

```
select nom from fournisseur
minus
select nom from fournisseur, mafourniture
 where fournisseur.f = mafourniture.f
```

Remarques

- <sélection1> et <sélection2> sont des requêtes select dont le nombre et le type des attributs sélectionnés doivent être identiques;
- l'instruction minus d'Oracle s'écrit: except pour la norme SQL. Il y a ici une incompatibilité, dûe au fait que cet opérateur n'appartient pas à SQL2 niveau 1;
- all, cité après union, évite l'élimination des doubles; en SQL2, all peut également être cité après intersect et except.

1. On trouve notamment *cross join* (produit cartésien), *natural join* (jointure naturelle), et *outer join* (jointure externe) qui est exprimée en Oracle par un ou deux signes + dans la clause where. Oracle couvre cependant toutes les possibilités d'expression de jointures de SQL2. Une jointure externe , qui est en général une équi-jointure, peut conserver les n-uplets des deux tables, même les n-uplets d'une table qui ne trouvent pas de correspondant dans l'autre table, des valeurs nulles remplaçant alors les informations qui auraient permis la jointure. Une jointure externe peut préserver les n-uplets sans correspondant venant de droite, de gauche ou des deux côtés (respectivement *right outer join*, *left outer join*, *full outer join*).

3.3.3 Renommage

Il est permis de renommer les relations.

Soit la relation employé (nom, directeur, salaire). L'interrogation

```
select emp.nom
 from employé emp, employé dir
  where emp.directeur = dir.nom and emp.salaire > dir.salaire
```

détermine le nom des employés qui gagnent plus que leur directeur. En SQL2, le renommage est introduit par as et il faut écrire:

```
from employé as emp, employé as dir
```

La clause where peut contenir une sélection, appelée sous-interrogation. L''interrogation suivante est une variante de l'interrogation précédente:

```
select emp.nom
  from employé emp
    where emp.salaire > (select dir.salaire
       from employé dir
         where dir.nom = emp.dir)
```

3.3.4 Prédicats

Pour écrire des prédicats, on dispose notamment de manière standard:

- des opérateurs de comparaison habituels >, <, >=, <=, =, <> (Oracle admet aussi ^= et != pour ce dernier opérateur),
- des opérateurs logiques and, or, not,
- des prédicats between, not between, null, not null
- de fonctions sur les chaînes de caractères:
  concaténation (||), sous-chaîne (substr), opérateur like, avec le caractère % pour une chaîne quelconque de 0 à n caractères, et le caractère _ pour un caractère et un seul,
- d'opérateurs ensemblistes: in, not in, some, any, all, exists, not exists.

L'interrogation

```
select nom
 from fournisseur
   where remise >= all (select remise from fournisseur)
```

donne le nom du(des) fournisseur(s) consentant la plus forte remise.

3.3.5 Fonctions

Pour faire des calculs, outre les opérateurs arithmétiques (+,-,*,/) qui peuvent figurer dans la partie select comme dans le prédicat, on dispose de fonctions s'appliquant à des groupes de n-uplets. On peut citer, notamment:

- des fonctions qui s'appliquent à des ensembles de valeur numériques: avg, sum, min, max,
- une fonction qui s'applique à n'importe quel ensemble: count (pour obtenir le nombre d'éléments).

Exemples
Soit la relation employé (nom, salaire_mensuel, prime_annuelle).

L'interrogation

```
select nom, (salaire_mensuel * 12) + prime_annuelle revenu
 from employé
  where salaire_mensuel * 12 + prime_annuelle > 100000
```

fournit le nom et le revenu annuel des employés ayant un revenu supérieur à 100000 francs. Un renommage d'attribut comme ici pour revenu est précédé de as dans la norme SQL2; on y écrit: as revenu.

L'interrogation

```
select count(*)
 from fournisseur
  where remise > (select avg(remise) from fournisseur)
```

donne le nombre de fournisseurs accordant une remise supérieure à la moyenne.

Remarque sur l'élimination des doubles
Quand on applique une fonction, il importe de savoir si l'on veut ou non conserver les doubles. C'est la spécification all qui s'applique par défaut. Si l'on désire l'élimination des doubles, il faut le spécifier par le mot-clé distinct.

3.3.6 Requêtes avec partitionnement

On considère une interrogation de la forme:

```
select a1, ..., ap
 from R1, ..., Rn
 [ where B1 ]
    group by b1, ..., bq
    [ having B2 ]
```

La clause group by permet de réaliser des agrégats. Un agrégat est un partitionnement horizontal d'une table en sous-tables en fonction des valeurs d'un ou de plusieurs attributs de partitionnement, suivi de l'application d'une ou plusieurs fonctions de calcul.

Ainsi, l'interrogation

```
select f, count (p), sum (qte)
 from mafourniture
  group by f
```

donne, grâce à un regroupement des n-uplets par fournisseur, pour chaque fournisseur, le nombre de produits fournis et la somme des quantités fournies (tous produits confondus).

L'interrogation

```
select nom
 from fournisseur
    group by nom
       having count (*) >= 2
```

délivre, du fait de la condition imposée par la clause having sur les groupes sélectionnés, les noms de fournisseur présents au moins deux fois dans la relation (cette requête permet de déceler les homonymes).

3.3.7 Division

L'interrogation suivante illustre une manière d'exprimer l'opération de division de l'algèbre relationnelle:

```
select f
from mafourniture M, produit P
  where M.p = P.p and P.origine = 'Dijon'
      group by f
       having count (*) = (select count (*) from produit  where origine = 'Dijon')
```

donne les numéros des fournisseurs qui fournissent au moins tous les produits originaires de Dijon.

Le group by f indique qu'un regroupement des n-uplets sélectionnés est effectué pour former ainsi divers ensembles:

```
E1    f1  p1  1 cassis   ...Dijon
      f1  p4  1 moutarde ...Dijon
      f1  p6  2 cornichon ...Dijon
E2    f2  p4  1 moutarde ...Dijon
      ...
```

Le having spécifie un prédicat qui doit être vrai pour que l'ensemble correspondant soit retenu dans la sélection. Ici, seul l'ensemble E1 satisfait le prédicat correspondant à un comptage.

3.3.8 Edition de résultats

Nous avons appelé instructions les actions définies en SQL et reconnues par au moins un groupe de normalisation (SQL2 ou ses extensions); nous appelons ordres les actions propres à l'interface SQL*PLUS d'Oracle.

Pour plus d'informations sur les ordres et instructions, l'ordre help:

help <instruction_ou_ordre>

est à votre disposition.

Un commentaire dans un script SQL*PLUS est introduit par:

- soit rem (pour remark) en début de ligne
- soit /* < commentaire> */

Oracle offre de multiples possibilités pour la présentation des résultats et la génération d'états. Nous donnons ici un seul exemple illustrant la possibilité de trier les résultats selon un critère et d'effectuer des calculs sur des groupes de n-uplets. La fabrication du script de cet exemple est présentée de manière progressive.

a) ce que l'on désire

On veut éditer la liste des produits fournis par les fournisseurs et les quantités fournies, triée selon le nom des fournisseurs et le nom des produits avec affichage du cumul des quantités (tous produits confondus) par fournisseur.
La requête permettant d'obtenir la liste des produits fournis par les fournisseurs et les quantités fournies est:

```
select nom, nom_p, qte
from fournisseur F, mafourniture M, produit P
  where F.f = M.f and M.p = P.p;
```

b) rupture et tri

Pour organiser cet état en groupes, chaque groupe correspondant à un fournisseur, on utilise l'ordre break suivant:

```
break on nom            /* définit une rupture sur le nom de fournisseur */
```

Ce qui nécessite d'ajoindre à la requête select précédente une clause order by:

```
order by nom, nom_p     /* tri selon le nom des fournisseurs, et en second lieu
                           selon le nom des produits */
```

c) calcul des sous-totaux

Pour obtenir pour chaque groupe, c'est à dire pour chaque fournisseur, le cumul des quantités, on utilise l'ordre compute suivant:

```
compute sum of qte on nom    /* à chaque rupture sur le nom de fournisseur, un
                                cumul des quantités sera fourni */
```

d) fignolage de la présentation

On peut utiliser divers ordres pour améliorer la présentation de l'état. On peut, par exemple:
- prévoir un titre de haut et de bas de page (ordres ttitle et btitle),
- renommer les colonnes de la requête (ordre column),
- sauter une ligne après chaque groupe (clause skip).

e) finalement le script désiré est le suivant:

```
break on nom skip 1
rem une rupture est définie sur le nom du fournisseur, avec saut d'une ligne
column nom_p heading 'nom du |produit'
```

```
rem la barre verticale fait que 'produit' est mis sous 'nom du'
ttitle 'liste des fournisseurs et de leurs produits'
rem ttitle = top title (titre en haut de page)
btitle 'rapport mensuel'
rem btitle = bottom title (titre en bas de page)
compute sum of qte on nom
rem il y a cumul des quantités à chaque changement de nom
select nom, nom_p, qte
from fournisseur F, mafourniture M, produit P
  where F.f = M.f and M.p = P.p
    order by nom, nom_p;
rem annulation des ordres de présentation, des ruptures et calculs associés
column nom_p clear
ttitle off
btitle off
clear breaks
clear computes
```

3.4 Mises à jour et changement de structure

3.4.1 Opérations de mise à jour

3.4.1.1 Insertion

L'insertion d'un n-uplet dans une relation se fait par l'instruction insert into.

Exemples:

```
insert into fournisseur values ('f8','Charpentier',6,'Marseille')
insert into mafourniture values ('f8','p2',5)
```

3.4.1.2 Destruction

La destruction de n-uplets dans une relation se fait par l'instruction delete.

Exemple

```
delete from fournisseur
 where remise < 2
```

3.4.1.3 Modification

La modification de valeurs figurant dans le champ d'une relation se fait par l'instruction update.

Exemple

Soit la relation employé (nom, salaire_mensuel, prime_annuelle)

```
update employé
set salaire_mensuel = salaire_mensuel + 200,
      prime_annuelle = prime_annuelle * 1.1
  where nom <> 'Tropayais'
```

revalorise le revenu des employés.

3.4.2 Contrôle des transactions et concurrence d'accès

Ce paragraphe contient seulement quelques éléments relatifs au partage des données entre utilisateurs. Les problèmes liés à la concurrence d'accès sont repris et développés au chapitre 5.

Lorque l'on effectue des mises à jour (par insert, delete ou update) sur des relations, les changements apportés ne deviennent effectifs qu'après exécution de l'instruction commit (que l'on pourrait traduire en français par *confirmer transaction* ou *rendre permanent*).

On peut ainsi définir une transaction comme une unité atomique de traitement, constituant un tout, au regard des pannes éventuelles et des accès multiples. Une transaction est l'ensemble des traitements effectués entre deux opérations commit.

Tant que les modifications proposées n'ont pas été rendues effectives par l'instruction commit, on peut les annuler en exécutant l'instruction rollback.

SQL offre la possibilité de partager des données entre plusieurs utilisateurs, sous réserve, bien sûr que le propriétaire des données ait octroyé à ces usagers les droits d'accès voulus.

L'intégrité des données lors d'accès multiples est garantie grâce à des verrous, posés explicitement ou implicitement, au niveau relation ou au niveau n-uplet.

Le principe de base est de permettre le maximum de parallélisme, mais en n'autorisant qu'un seul utilisateur à la fois à travailler en mise à jour sur un n-uplet donné.

Ainsi, toute instruction de mise à jour (update, delete ou insert) pose implicitement un verrou sur les n-uplets concernés. Les n-uplets ne sont déverrouillés qu'après exécution de l'instruction commit. Pendant toute la durée de la transaction, les autres utilisateurs peuvent travailler, en lecture, sur l'état avant mises à jour des n-uplets verrouillés. Par défaut, ce verrouillage est fin puisqu'il autorise des mises à jour simultanées de n-uplets distincts d'une même relation.

3.4.3 Restructuration et suppression de table

SQL permet, pour toute table existante :

- d'ajouter un nouvel attribut

```
alter table mafourniture
add montant decimal(8,2)
```

- de modifier la déclaration d'un attribut

```
alter table produit
modify column nom_p varchar(20)
```

La taille de l'attribut nom_p a été changée. Le verbe modify est remplacé par

alter en SQL2;

• de supprimer un attribut, en SQL2 (mais pas en Oracle).

SQL permet de détruire une table (ou une vue, introduite au paragraphe suivant). L'instruction:

```
drop table mafourniture
```

fait disparaitre tous les n-uplets de cette table, comme:

```
delete from mafourniture
```

mais, en plus, supprime la déclaration de la table, c'est-à-dire la structure d'accueil des données.

3.5 Vue et confidentialité

3.5.1 Notion de vue

SQL permet de définir des *vues*, c'est à dire « diverses manières de voir » des données figurant dans une base. Les vues sont des relations virtuelles, qui ne contiennent aucune donnée en propre et que l'on peut manipuler comme des relations réelles.

Cette notion permet, par exemple, de mettre en oeuvre, associé à une application donnée, un sous-modèle du modèle principal de la base de données.

Exemples

```
create view fourparis
as select f, nom, remise
      from fournisseur
        where ville = 'Paris'
```

crée la relation (virtuelle) regroupant les fournisseurs de Paris.

```
create view mafourparis (f,p,qte)
as select M.f, M.p, M.qte
     from mafourniture M, fourparis F
       where M.f = F.f
```

crée une vue de mafourniture restreinte aux fournisseurs parisiens.

```
select distinct nom_p
 from mafourparis, produit
   where produit.p = mafourparis.p
```

donne le nom des produits fournis par les fournisseurs parisiens.

Remarques

- la notion de vue permet de simplifier l'accès aux données, ainsi que l'illustre l'exemple ci-dessus;
- elle est un facteur pour assurer l'indépendance des données. Ainsi, les programmes doivent plutôt utiliser des vues comportant les seules données dont ils ont besoin. Ceci favorise le fait que les modifications de la structure et la nature des relations entre les tables puissent rester invisibles des programmes. A noter cependant que certaines vues (résultant de jointure de tables, par exemple) ne peuvent être utilisées pour une mise à jour;
- en outre, elle permet de mieux garantir la confidentialité des données.

3.5.2 Confidentialité

Nous sommes le propriétaire de toute table ou vue que nous créons. Par défaut, les données sont privées, donc réservées à leur propriétaire. SQL permet:

- d'accorder (grant) des privilèges à d'autres utilisateurs sur nos tables et nos vues, ou sur celles pour lesquelles nous avons reçu des privilèges avec transmission possible (with grant option)
- de retirer (revoke) des privilèges que nous avons accordés à d'autres, ainsi que les privilèges éventuellement transmis par ceux à qui nous les retirons.

L'instruction grant comporte quatre clauses de base:

grant accorder un privilège
on sur une table ou une vue
to à un utilisateur, ou à un groupe d'utilisateurs ou à tous (public)
with grant option transmission possible des privilèges reçus (facultative)

Exemples

```
grant select, insert, update on produit to dupond
```

Les autres privilèges communs à SQL2 et Oracle sont: delete, references. La liste de tous les privilèges est remplacée par all.

La suppression des privilèges accordés se fait, par exemple, par:

```
revoke all on produit from dupond
```

L'instruction

```
grant select on mafourparis to public
```

donne accès, en lecture uniquement, aux trois attributs de cette vue concernant les fournisseurs de Paris. Le nombre d'attributs de la vue pourrait être différent de trois.

3.6 Exemples d'interrogations

Nous reprenons ici les mêmes questions que celles qui ont été vues au § 2.3.4, pour comparer les requêtes en algèbre relationnelle et en SQL.

Q1: Trouver le numéro des fournisseurs qui me fournissent au moins un article

```
select distinct f from mafourniture
```

Q2: Trouver le numéro des fournisseurs qui ne me fournissent aucun article

```
rem on peut utiliser minus , <>  all , not in
select f from fournisseur
where f not in (select f from mafourniture)
```

Q3: Trouver le numéro des fournisseurs qui me fournissent p6

```
select distinct f from mafourniture
where p = 'p6'
```

Q4: Trouver le numéro des fournisseurs qui me fournissent quelque chose d'autre que p6

```
select distinct f from mafourniture
where p <> 'p6'
```

Q4': Trouver le numéro des fournisseurs qui me fournissent quelque chose, mais pas p6

```
select f from mafourniture
minus
select f from mafourniture where p = 'p6'
```

Q5: Trouver le numéro des fournisseurs qui ne fournissent que p6

```
select f from mafourniture
minus
select f from mafourniture
where p <> 'p6'
```

Q6: Trouver le numéro des fournisseurs qui me fournissent chacun au moins p4, p5 et p6

```
select f from mafourniture
where p='p6'
intersect
select f from  mafourniture
where p='p4'
intersect
select f from mafourniture
where p='p5'
```

Q7: Trouver le nom et la ville des fournisseurs qui me fournissent tous les produits originaires de Dijon

Il s'agit ici d'une question voisine d'une question traitée précédemment. L'utilisation, facultative, d'une vue comme prodijon évite de répéter deux fois la même sélection.

```
create view prodijon
as select p from produit where origine='Dijon';

create view bon as
select f from mafourniture
where p in (select p from prodijon)
group by f
having count(*) =
(select count(*) from prodijon);

select distinct nom, ville
from fournisseur
where f in (select f from bon);
drop view bon;
drop view prodijon;
```

Q8: Trouver les numéros des fournisseurs qui me fournissent au moins deux produits.

```
select f
from mafourniture
group by f
having count(*) >= 2
```

3.7 Récapitulatif des instructions présentées

3.7.1 Définition des données

create table, create view
alter table
drop table, drop view

3.7.2 Manipulation des données

select, union…
insert, delete, update

3.7.3 Exploitation de la base

grant, revoke
commit, rollback
set transaction

Quelques autres instructions SQL sont présentées au chapitre 5.

CHAPITRE 4

Conception de schémas relationnels

4.1 Exemple introductif

4.1.1 Conception de schémas relationnels

Le concepteur d'une base de données doit, à partir d'une réalité qui lui est imposée, proposer un *bon* système de relations pour la représenter. La qualité des choix faits à ce niveau conditionne largement le succès de l'informatisation et on aimerait pouvoir s'appuyer sur des modèles et des méthodes éprouvés. La citation suivante de Claude DELOBEL et Michel ADIBA pose le problème:

« Le fait d'aboutir à un schéma relationnel est-il le résultat d'une démarche expérimentale effectuée à partir de l'analyse des besoins que doit satisfaire la base de données ou bien est-il possible de proposer une démarche conduisant à une construction plus ou moins automatique ? La réponse à cette question n'est pas simple et, dans l'état actuel de nos connaissances, on peut dire que jusqu'à présent le processus de conception a principalement résulté d'une démarche itérative associant l'expérience, la logique, les succès et les erreurs. »

Nous partageons cette opinion et la conception de bons schémas relationnels nécessite avant tout de l'expérience et du bon sens ; cependant, comme d'habitude, la connaissance de modèles et de méthodes peut largement aider le concepteur. Nous présentons donc, dans ce chapitre, un modèle d'aide à la conception de schémas relationnels. Introduit par CODD, il repose sur la notion de *formes normales*.

Avant d'aborder la présentation de ce modèle, nous discutons de ce que doit être un bon système de relations.

4.1.2 Anomalies et redondances

Etant donné des attributs, les choix possibles, pour un système de relations sont nombreux. Aux deux extrémités de l'échelle, il est toujours possible d'adopter:

- une seule relation ; la relation unique qui décrit la base est appelée relation universelle,
- une collection de relations binaires.

À titre d'exercice, le lecteur pourra considérer la relation universelle de la base épicerie et imaginer les relations binaires qu'il est possible d'adopter. À l'évidence, dans ce cas, les deux choix extrêmes sont mauvais.

Considérons l'exemple plus simple suivant:

Base transport
On dispose de trois attributs:

- f (fournisseur),
- ville (adresse du fournisseur)
- frais (de transport)

et de deux contraintes d'intégrité:

- I1: Un fournisseur donné n'a qu'une seule adresse,
- I2: Les frais de transport ne dépendent que de la ville du fournisseur (état des communications) ; en conséquence, les frais associés aux fournisseurs d'une même ville sont identiques.

On appelle R la relation universelle associée à la base transport et on considère le contenu suivant qui vérifie I1 et I2.

f	ville	frais
f1	Paris	100
f2	Paris	100
f3	Rennes	50
f4	Redon	50

L'adoption de la relation R (f, ville, frais) pour la base transport induit des *redondances* d'informations et des risques d'*anomalies* lors des modifications de la base (ces deux points, redondances et anomalies, sont d'ailleurs intimement liés).

Redondance d'informations
Sur l'exemple précédent on constate que l'association des frais 100 à la ville Paris est stockée deux fois.

Anomalies de modification

Si *f1 déménage pour Redon*, l'opération de modification suppose l'acquisition préalable des frais associés à Redon: une mise à jour « en aveugle » (ie. qui n'utilise que l'information explicitée dans la phrase *f1 déménage pour Redon*) entraîne un contenu incohérent pour R.

Si les frais associés à Paris changent, de nombreux n-uplets (2 dans l'exemple !) doivent être modifiés.

Anomalies de destruction

On peut illustrer une anomalie provoquée par une destruction en considérant, à partir du contenu de l'exemple, l'exécution des trois requêtes suivantes:

1. Quels sont les frais associés à Redon ?

• Réponse: 50

2. Détruire le fournisseur f4.

• Réponse: destruction effectuée

3. Quels sont les frais associés à Redon ?

• Réponse: information inconnue

Ce comportement est anormal dans la mesure où l'utilisateur n'est pas prévenu des conséquences de sa destruction (en fait, même s'il était prévenu, il n'est pas évident qu'un tel comportement le satisfasse).

Anomalies d'insertion

Les anomalies d'insertion sont duales des anomalies de destruction: il n'est pas directement possible d'introduire dans la base l'information *les frais associés à Brest ont pour valeur 75*.

4.2 Normalisation

4.2.1 Décomposer un schéma relationnel

Nous venons de voir que la relation R (f, ville, frais) n'était pas un bon choix pour la base transport. Pour proposer d'autres solutions, il faut « casser », on dit décomposer, la relation R en plusieurs autres relations. La base transport est suffisamment simple pour qu'on trouve facilement la bonne décomposition, toutefois, pour mieux comprendre les concepts mis en cause, nous en essayons trois.

Première décomposition
R1

f	frais
f1	100
f2	100
f3	50
f4	50

R2

ville	frais
Paris	100
Rennes	50
Redon	50

Cette décomposition est particulièrement rédhibitoire: la question *Quelle est la ville de f4 ?* qui ne peut s'exprimer que par $(R_1 * R_2)\{f = 'f4'\}$ [ville] fournit en résultat Rennes et Redon ! En d'autres termes, il n'y a pas les mêmes informations dans R et dans le couple (R_1, R_2).

Deuxième décomposition
R'1

f	ville
f1	Paris
f2	Paris
f3	Rennes
f4	Redon

R'2

f	frais
f1	100
f2	100
f3	50

f	frais
f4	50

L'inconvénient majeur de la première décomposition a disparu: la question *Quelle est la ville de f4 ?* s'exprime par R'_1 {f = 'f4'} [ville] et délivre le bon résultat. En fait l'information disponible dans un contenu légal de R est préservée dans le couple (R'_1, R'_2).

En revanche les anomalies de mise à jour citées pour R subsistent.

Troisième décomposition
R''1

f	ville
f1	Paris
f2	Paris
f3	Rennes
f4	Redon

R''2

ville	frais
Paris	100
Rennes	50
Redon	50

Sur cette troisième décomposition on constate que l'information est conservée et que les anomalies citées disparaissent.

Ces trois décompositions posent donc la question de savoir ce qui les distinguent: c'est, à l'évidence, l'existence des deux contraintes d'intégrité I1 et I2. A partir de cette remarque nous pouvons présenter le plan de la suite de ce paragraphe:

1. Définir un langage d'expression de contraintes d'intégrité,
2. Définir plus formellement l'opération de décomposition et, en particulier, préciser ses rapports avec les contraintes d'intégrité,
3. Proposer des critères de qualité pour ce qu'on appelle un schéma relationnel,
4. Donner des algorithmes pour produire des décompositions satisfaisant les critères précédents.

4.2.2 Dépendances fonctionnelles

4.2.2.1 Définition et exemples

La notion de dépendance fonctionnelle est un cas particulier simple de contrainte d'intégrité.

Définition 4.1 On considère une relation R(U) ; on dit que Y *dépend fonctionnellement* de X, ce qu'on note par $X \rightarrow Y$, si et seulement si:

$\forall$ r légal, <x, y, z> $\in$ r $\wedge$ <x, y', z'> $\in$ r $\Rightarrow$ y = y'

En d'autres termes, deux lignes égales sur X doivent l'être également sur Y, ou, de manière plus imagée « quand on connaît X on connaît Y ».

On dira indifféremment : Y dépend fonctionnellement de X ou X détermine Y.

Exemple 1: La base transport.
Les contraintes I1 et I2 citées plus haut peuvent être traduites par:

I1 : f $\rightarrow$ ville
I2 : ville $\rightarrow$ frais

Exemple 2: La base épicerie.
Sur la base épicerie, on a évidemment:

f $\rightarrow$ nom remise ville
p $\rightarrow$ nom_p couleur origine
f p $\rightarrow$ qte

Nous pouvons introduire la notion de schéma relationnel.

Définition 4.2 Un *schéma relationnel* est un ensemble de couples <R(U), F> où F est un ensemble de dépendances fonctionnelles portant sur les attributs U.

Au niveau des notations, nous omettrons fréquemment le nom des relations dans un schéma (deux schémas qui ne diffèrent que sur les noms des relations sont évidemment équivalents. Lorsque l'ensemble des éléments d'un schéma est réduit à un élément, nous le noterons simplement <R(U), F> ou <U,F>. Les éléments constitutifs d'un schéma seront appelés sous-schémas.

Le problème posé par la conception d'un bon système de relations passe par un stade où on dispose de U, l'ensemble des attributs de la base et de F, l'ensemble des contraintes d'intégrité: il s'agit du schéma <U, F> correspondant à la relation universelle. Le but est alors de trouver, par décomposition, un schéma plus satisfaisant.

4.2.2.2 Fermeture d'un ensemble de DF

On dispose donc d'un moyen d'imposer à un ensemble d'attributs U des contraintes d'intégrité: il suffit de donner un ensemble F de dépendances fonctionnelles. Une question se pose alors: les contraintes imposées par F se

limitent-elles à l'énumération des éléments de F ? La réponse est évidemment non, comme en témoigne l'exemple 1 précédent:

Schéma: <R (f, ville, frais), {f → ville, ville → frais}>
Si r est un contenu légal de R et si deux lignes sont égales sur f, elles le sont évidemment sur frais: r respecte donc la contrainte f → frais

Il y a donc lieu de distinguer les dépendances désignées (explicitées …) F des dépendances réellement imposées F^+. On appelle F^+ la *fermeture* de F et les éléments de F^+ sont les dépendances *logiquement impliquées* par F - ce sont les dépendances vérifiées par tout contenu r de R qui vérifie F.

4.2.2.3 Retour sur la notion de clé

On peut, à l'aide de la notion de dépendance fonctionnelle, redéfinir la notion de clé:

Définition 4.3 Une *clé* pour un schéma <R(U), F> est un ensemble minimum d'attributs X tel que $X \rightarrow U \in F^+$.

Tout schéma admet au moins une clé puisque, pour tout F, U → U. En revanche, l'unicité n'est pas obligatoire comme en témoigne l'exemple suivant:

Exemple 3: La base La Poste
<R (adresse, codepostal, burdist),
{codepostal → burdist, adresse burdist → codepostal}>

(35200 est un code postal de Rennes, mais à partir de Rennes il faut l'adresse pour savoir si le code est 35000, 35200 …). Ce schéma admet deux clés: adresse codepostal et adresse burdist

On a vu que les SGBD relationnels permettent de *désigner* une clé (au moins) par relation ; le système se charge au minimum de contrôler la validité des modifications par rapport à ces clés. On va donc chercher à ce que les clés désignées des relations captent le plus grand nombre possible des contraintes d'intégrité qu'on souhaite imposer. A titre d'exemple, reprenons la base transport et les trois décompositions que nous avons données:

Base transport:
<R (f, ville, frais), {I1 : f → ville, I2 : ville → frais}>

Première décomposition
R1(f, frais) et R2(ville, frais), I1 n'est pas captée par les clés

Deuxième décomposition
R'1(f, ville) et R'2(f, frais), I2 n'est pas captée par les clés

Troisième décomposition
R''1(f, ville) et R''2(ville, frais), I1 et I2 sont captées par les clés

Nous éclaircirons les rapports entre qualité d'un schéma relationnel et clés au cours des paragraphes suivants.

4.2.2.4 Détermination des implications logiques d'un ensemble de DF

Puisque le concepteur d'un schéma explicite un F et impose en réalité F^+, il parait intéressant de pouvoir cerner ce qu'est exactement F^+. Son calcul exhaustif est par nature long ; en effet, si U = A1 … An, on sait que F^+ contient au moins toutes les dépendances de la forme $X \rightarrow Y$ avec $Y \subseteq X$ (on appelle ces dépendances des dépendances triviales - elles sont nombreuses !) et ceci quel que soit F.

Pour cerner F^+, nous donnons un algorithme qui, à partir d'un X donné, calcule X^+, l'ensemble des Y tels que $X \rightarrow Y \in F^+$.

X_{vieux} := X ;
jqa CVG faire
    X_{neuf} := $X_{vieux} \cup$
        { A | $Y \rightarrow Z \in F, A \in Z \wedge Y \subseteq X_{vieux}$ };
    qd $X_{neuf} = X_{vieux}$ sortirpar CVG ;
    X_{vieux} := X_{neuf} ;
fait ;
CVG :X^+ := X_{neuf} ;
fiter

Exemple
X = BD
F = { AB → C, (1) D → EG (5),
    C → A, (2) BE → C (6),
    BC → D, (3) CG → BD (7),
    ACD → B, (4) CE → AG (8) }

Pas	X_{vieux}	Dépendances applicables (et non encore utilisées)	Xneuf
1	BD	(5) D → EG	BDEG
2	BDEG	(6) BE → C	BCDEG
3	BCDEG	(2) C → A (3) BC → D (7) CG → BD (8) CE → AG	ABCDEG
4	ABCDEG	(1) AB → C (4) ACD → B	ABCDEG

4.2.2.5 Axiomes d'ARMSTRONG

Soit U un ensemble d'attributs et F un ensemble de dépendances fonctionnelles. Les axiomes d'ARMSTRONG sont les suivants:

- A1:$X \subseteq Y \Rightarrow Y \rightarrow X$ (réflexivité)
- A2:$X \rightarrow Y \Rightarrow XZ \rightarrow YZ$ (augmentation)
- A3:$X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$ (transitivité)

La propriété suivante (dont la démonstration est laissée au lecteur) donne un autre moyen de cerner F^+.

Propriété 4.1 $X \rightarrow Y \in F^+$ si et seulement si $X \rightarrow Y$ peut être établi à partir de F en appliquant un nombre fini de fois un des trois axiomes d'ARMSTRONG.

4.2.2.6 Couverture minimale

Définition 4.4 On dit que deux ensembles de dépendances fonctionnelles F et G sont *équivalents* s'ils imposent exactement les mêmes contraintes, c'est-à-dire si et seulement si $F^+ = G^+$.

Étant donné un F il existe peut-être un G équivalent plus lisible ou plus manipulable ; la notion de *couverture minimale*, qui est utilisée au § 4.252 a été introduite à cette fin.

Définition 4.5 On appelle *couverture minimale* de F un ensemble G équivalent à F qui vérifie les trois propriétés suivantes:

1. Toutes les parties droites sont réduites à un seul élément,
2. Aucune partie gauche ne contient d'élément redondant:
   $\forall X \rightarrow A \in G$ et $Z \subset X$, $(G - \{X \rightarrow A\} \cup \{Z \rightarrow A\})^+ \neq G^+$
3. Il n'y a pas de dépendance superflue:
   $\forall X \rightarrow A \in G$, $(G - \{X \rightarrow A\})^+ \neq G^+$

Un algorithme de calcul d'une (il y en a toujours au moins une) couverture minimale consiste à « imposer », dans l'ordre, ces trois conditions:

- Pas 1: On expanse les dépendances $X \rightarrow A_1 \ldots A_n$ en $X \rightarrow A_1$, ..., $X \rightarrow A_n$. On obtient F'.
- Pas 2: Pour chaque partie gauche de F', on choisit un ordre d'examen des attributs et on cherche à les enlever. Les suppressions successives conduisent à F''.
- Pas 3: On choisit un ordre d'examen de F'' et on cherche à enlever chacune des dépendances considérées. Les suppressions successives conduisent à G, le résultat final.

Remarque

On utilise parfois, pour qualifier une dépendance fonctionnelle dont la partie gauche ne contient pas d'élément redondant, l'expression *dépendance élémentaire*.

4.2.3 Décompositions

4.2.3.1 Définitions

On considère un ensemble d'attributs U et $\delta = \{U_1, ..., U_n\}$ avec $U = U_1 \cup ... \cup U_n$; on va se servir de δ pour décomposer un schéma $S = <R(U), F>$ en plusieurs schémas $<R_1 (U_1), F_1>, ..., <R_n (U_n), F_n>$. On dit que δ est une décomposition de S.

Il faut pouvoir caractériser le contenu d'une relation $R_i(U_i)$ à partir du contenu de R et caractériser F_i à partir de F. En ce qui concerne les contenus, il semble naturel de poser $R_i(U_i) = R[U_i]$. Pour les dépendances, on introduit la notion de projection des dépendances:

Définition 4.6 La *projection* F[U] d'un ensemble de dépendances fonctionnelles F sur un ensemble d'attributs U est définie par:

$$F[U] = \{X \rightarrow Y \in F^+ \mid X \subseteq U, Y \subseteq U\}$$

C'est donc l'ensemble des dépendances fonctionnelles de F^+ (et non seulement de F) dont les parties gauches et droites sont dans U. Ainsi $\{A \rightarrow B, B \rightarrow C\}[AC]$ contient $A \rightarrow C$.

Lorsqu'on considère une décomposition δ, on adopte naturellement $F_i = F[U_i]$.

Exemple: La base transport

$S = <R (f, ville, frais), \{f \rightarrow ville, ville \rightarrow frais\}>$

Les trois décompositions citées précédemment sont:

$\delta = \{f\ frais, ville\ frais\} <R_1(f, frais), \{f \rightarrow frais\}>$ et $<R_2(ville, frais), \{ville \rightarrow frais\}>$
$\delta' = \{f\ ville, f\ frais\} <R'_1(f, ville), \{f \rightarrow ville\}>$ et $<R'_2(f, frais), \{f \rightarrow frais\}>$
$\delta'' = \{f\ ville, ville\ frais\} <R''_1(f, ville), \{f \rightarrow ville\} >$ et $<R''_2(ville, frais), \{ville \rightarrow frais\}>$

Définition 4.7 Une décomposition $\delta = \{U_1, ..., U_n\}$ est *valide* si et seulement si, pour tout contenu légal r de R on a:

$$r = r_1[U_1] * ... * r_n[U_n]$$

Définition 4.8 Une décomposition *préserve les dépendances* si et seulement si:

$$F^+ = \left(\bigcup_i F[U_i] \right)^+$$

Une décomposition raisonnable ne peut pas ne pas être valide. Sur la base transport, δ n'est pas valide ce qui conduit à son rejet sans appel. On peut à la rigueur (ce n'est pas souhaitable) vivre avec une décomposition qui ne préserve pas les dépendances: les dépendances « perdues » rejoignent le

stock des contraintes qu'on n'a pas su exprimer. Sur la base transport, δ' ne préserve pas les dépendances. Dans ce cas particulier la dépendance perdue explique les anomalies et il y a un meilleur choix, δ'', qui est valide, qui préserve les dépendances et qui, de plus, est exempte d'anomalies.

4.2.3.2 Critère de validité

Nous donnons, pour les décompositions en deux éléments, une condition nécessaire et suffisante de validité.[1]

Propriété 4.2 Une décomposition δ = {U1, U2} est valide si et seulement si:

- ou $U_1 \cap U_2 \rightarrow U_1 - U_2 \in F^+$ (1)
- ou $U_1 \cap U_2 \rightarrow U_2 - U_1 \in F^+$ (2)

Considérons un schéma <R(X, Y, Z), F>, et δ = {XY, XZ} avec donc $X = U_1 \cap U_2$, $Y = U_1 - U_2$, $Z = U_2 - U_1$, et deux n-uplets <x, y1, z1> et <x, y2, z2> d'un contenu légal pour R. Il y a un contenu légal r pour R qui ne contient que ces deux n-uplets.

Si δ est valide, r[XY] * r[XZ] = r. Donc les n-uplets <x, y1, z1>, <x, y1, z2>, <x, y2, z1> et <x, y2, z2> sont dans r, ce qui implique:

- ou y1 = y2
- ou z1 = z2

Donc, dans un contenu légal pour R, deux lignes égales sur X sont égales sur Y ou sur Z.

La réciproque ne pose pas de difficulté.

4.2.3.3 Validité, préservation des dépendances et anomalies de mise à jour

Il est indispensable qu'une décomposition soit valide et il est souhaitable qu'elle préserve les dépendances ; ces deux critères ne sont cependant pas, à eux seuls, suffisants pour assurer que la décomposition est exempte d'anomalie de mise à jour. On peut facilement s'en convaincre en considérant l'exemple suivant:

Base des stocks

On ajoute à la base transport deux attributs:

1. D'autres propriétés sur la décomposition sont énoncées et démontrées dans :
   FORET A., FRESNAIS P. , Évaluation comparative d'algorithmes de normalisation des schémas relationnels, article de la revue *Ingénierie des systèmes d'information* (*ISI*), Hermès Volume 3 - n° 4 / 1995

- p (produit fourni),
- qte (quantité en stock)

ce qui conduit au schéma:

<R (f, ville, frais, p, qte), {f → ville, ville → frais, f p → qte}>

La décomposition δ = {f p qte, f ville frais} produit les deux sous-schémas:

<R1(f, p, qte), {f p → qte}>
<R2(f, ville, frais), {f → ville, ville → frais, f → frais}>

δ est valide, puisque f → ville frais ∈ F^+ (cf. la CNS de validité donnée plus haut) et δ préserve les dépendances. Pourtant le sous-schéma R2 comporte des risques d'anomalies de mise à jour (cf. § 3.1). δ est donc une mauvaise décomposition.

Nous allons donc nous attacher à définir des critères – les formes normales – visant à diminuer le risque d'occurrences d'anomalies de mise à jour. Une fois ces critères définis, nous donnerons des algorithmes dont le but est de produire, par décomposition, des sous-schémas qui les vérifient.

4.2.4 Formes normales

4.2.4.1 Notions préliminaires

La notion de forme normale a été introduite par CODD qui en a proposé trois: 1NF, 2NF, 3NF. Le fait que certaines relations en 3NF présentent encore des anomalies a conduit BOYCE et CODD a proposer une autre forme normale, la BCNF. Comme les anomalies de mise à jour sont liées aux contraintes d'intégrité, la prise en compte de contraintes plus riches que les dépendances fonctionnelles (dépendances multivaluées, dépendances de jointure) ont également donné lieu à d'autres formes normales: 4NF …

L'idée de départ consiste à réduire, en imposant des contraintes, l'ensemble des relations admises. Cette réduction vise à diminuer le risque d'occurrences d'anomalies de mise à jour. On notera que ces réductions successives sont « concentriques »: toute relation en BCNF est en 3NF, toute relation en 3NF est en 2NF … Cette propriété laisse penser que seule la forme la plus contrainte (la BCNF quand on se limite aux dépendances fonctionnelles) mérite d'être présentée. En fait, les deux propriétés suivantes, que nous établirons plus loin, expliquent l'intérêt de la 3NF:

Propriété 4.3 Pour tout schéma de départ, il est toujours possible de trouver une décomposition valide qui préserve les dépendances produisant une collection de sous-schémas en 3NF.

Propriété 4.4 Il existe des schémas pour lesquels aucune des décompositions valides qui produisent une collection de sous-schémas en BCNF ne préserve les dépendances.

Avant de définir 3NF et BCNF, il convient de remarquer le rôle très particulier joué par les clés d'une relation: on enregistre dans la relation des associations entre une clé et les autres attributs. Les anomalies proviennent en général de l'existence d'associations qui « ne passent pas directement » par les clés. Ainsi, sur la base transport, la clé du schéma initial est *f* alors qu'il existe une association entre *ville* et *frais*. On remarque que c'est cette dépendance qui est à l'origine des anomalies constatées.

Définition 4.9 Attributs non primitifs et transitivité

1. On dit qu'un attribut est *non primitif* si et seulement s'il n'appartient à aucune clé.
2. On dit qu'un attribut A est *transitivement dépendant* de X si et seulement s'il existe Y tel que:
   $X \rightarrow Y$ et $Y \rightarrow A$, avec
   $A \notin Y$ et
   $\neg(Y \rightarrow X)$

4.2.4.2 La troisième forme normale

Lorsqu'un attribut non primitif est transitivement dépendant d'une clé il y a un risque d'anomalies de mise à jour que nous illustrons sur deux exemples.

a) Premier exemple: la base association

Une association propose à ses adhérents (a) un certains nombre de clubs (c) et la participation aux activités d'un club donné suppose le paiement d'une cotisation dont le montant (m) ne dépend que du club. La seule dépendance fonctionnelle est donc $c \rightarrow m$ et on considère le schéma initial suivant:

$<R (a, c, m), \{c \rightarrow m\}>$

La seule clé de ce schéma est donc ac et le seul attribut non primitif, m, est transitivement dépendant de la clé puisqu'on a:

$ac \rightarrow c \rightarrow m$ et $\neg(c \rightarrow ac)$

On constate la possibilité d'anomalies de mise à jour dont l'archétype est donné par la séquence:

1. Quel est le montant de la cotisation du club de méditation transcendantale?

Réponse:200 F

2. L'adhérent Tartempion quitte l'association.

Réponse: destruction effectuée ... mais si Tartempion le seul inscrit au club de méditation transcendantale:

3. Quel est le montant de la cotisation du club de méditation transcendantale?

Réponse:information inconnue

b) Deuxième exemple: la base transport

Si on reprend le schéma initial de la base transport:

$<R (f, ville, frais), \{f \rightarrow ville, ville \rightarrow frais\}>$

on constate que la seule clé est f. frais est non primitif et il dépend transitivement par ville de f ; le lecteur fera sans peine le lien entre cette transitivité et les anomalies exhibées au début de ce chapitre. La troisième décomposition δ'' (valide et préservant les dépendances) produit les deux sous-schémas $<R''_1$(f, ville), {f → ville}> et $<R''_2$(ville, frais), {ville → frais}> qui sont tous les deux en 3NF. On se souvient que ce schéma de base est exempt d'anomalies.

c) Interdire les transitivités

Il est clair, au vu des deux exemples précédents, que, si on interdit toute transitivité des clés vers les attributs non primitifs, on supprime certaines causes d'anomalies. C'est l'objectif visé par les deuxième et troisième formes normales. La deuxième forme normale interdit les transitivités du type de celle de la base association (elle passent par un sous-ensemble des clés) et la troisième étend cette interdiction aux transitivités strictes.

Définition 4.10 Une relation est en *troisième forme normale* (3NF) si et seulement si aucun attribut non primitif n'est transitivement dépendant d'une clé.

Remarque

Il existe toutefois des schémas en 3NF qui ne sont pas exempts d'anomalies, c'est le cas de la base La Poste :

< R (adresse, codepostal, burdist),
{codepostal → burdist,
adresse burdist → codepostal} >

Tous les attributs étant primitifs, la propriété définissant la 3NF est trivialement vérifiée. On peut constater, sur ce schéma, des anomalies du même type que celles que nous avons déjà rencontrées:

1. Quel est le bureau correspondant à 35410 ?
Réponse:Châteaugiron

2. Détruire l'adresse de Tartempion.
Réponse: destruction effectuée … mais si Tartempion était la seule adresse dépendant de 35410:

3. Quel est le bureau correspondant à 35410 ?
Réponse:information inconnue

En fait, sur la base La Poste, il y a une association entre burdist et codepostal ; dans la relation de départ, cette association ne peut pas être directement mémorisée puisque adresse appartient à toutes les clés. Cette constatation est à l'origine de la définition de la BCNF.

4.2.4.3 La forme normale de BOYCE-CODD

Définition 4.11 Un schéma est en forme normale de BOYCE-CODD (BCNF) si et seulement si:

$\forall X \rightarrow A \in F^+$ et $A \notin X$, X contient une clé.

En d'autre termes, une relation est en BCNF si toutes les dépendances non triviales sont captées par les clés.

Remarque 1: BCNF $\Rightarrow$ 3NF

Une relation en BCNF est nécessairement exempte de transitivité violant la définition de la 3NF: si $X \rightarrow Y$ et $Y \rightarrow A$ et si la relation est en BCNF, on sait que Y contient une clé ; on a donc $Y \rightarrow X$.

La décomposition δ'' de la base transport produit deux sous-schémas en BCNF. On a vu qu'ils sont également en 3NF.

Remarque 2: $\neg$ (3NF $\Rightarrow$ BCNF)!

Il suffit de constater que la base La Poste, en 3NF, n'est pas en BCNF.

Le fait que cette base ne soit pas en BCNF permet de « mettre le doigt où le bât blesse », c'est-à-dire sur la dépendance codepostal $\rightarrow$ burdist En revanche, on pourra constater sur cet exemple que toute décomposition valide visant à produire des relations en BCNF provoque la perte de dépendance. Cet exemple, typiquement « retors », illustre qu'il existe des cas où il n'y a pas de stratégie totalement gagnante ; on a le choix entre:

1. vivre avec une relation en 3NF, tout en sachant qu'il faudra explicitement intervenir (dans les programmes d'application) pour assurer l'absence d'anomalies, ou
2. vivre avec deux relations en BCNF, par exemple R1(**codepostal**, burdist) et R2(**adresse**, codepostal) en contrôlant explicitement, dans les programmes d'application, la dépendance adresse burdist $\rightarrow$ codepostal.

On remarque cependant que, dans ce cas particulier, le profil de l'application permet de choisir facilement la deuxième solution: la relation R1, dont le contenu ne dépend que de la Poste ne bouge presque jamais et on peut admettre que le programme d'application associé à une telle modification (par exemple la création du 35800 à Rennes) implique un « lavage » soigné du carnet d'adresse (R2). En revanche, lors de la saisie d'une adresse, la vérification de la cohérence du triplet fourni n'est pas trop compliquée.

4.2.5 Algorithmes de décomposition

4.2.5.1 Décomposition valide en BCNF

Principe

Pour obtenir une décomposition valide avec des sous-schémas en BCNF, on part d'un schéma initial $<U_0, F_0>$. S'il n'est pas en BCNF on va l'éclater en deux sous-schémas puis on va itérer le processus sur chacun des sous-schémas obtenus. On établit donc un arbre binaire de décompositions successives dont la validité est assurée par la CNS donnée plus haut. La diminution stricte du nombre d'attributs d'un pas sur l'autre assure la terminaison sur chacune des branches.

Algorithme de décomposition en BCNF

Entrée

$<U_0, F_0>$, schéma initial

Structure de données

X, ensemble de couples <U, F>

Sortie

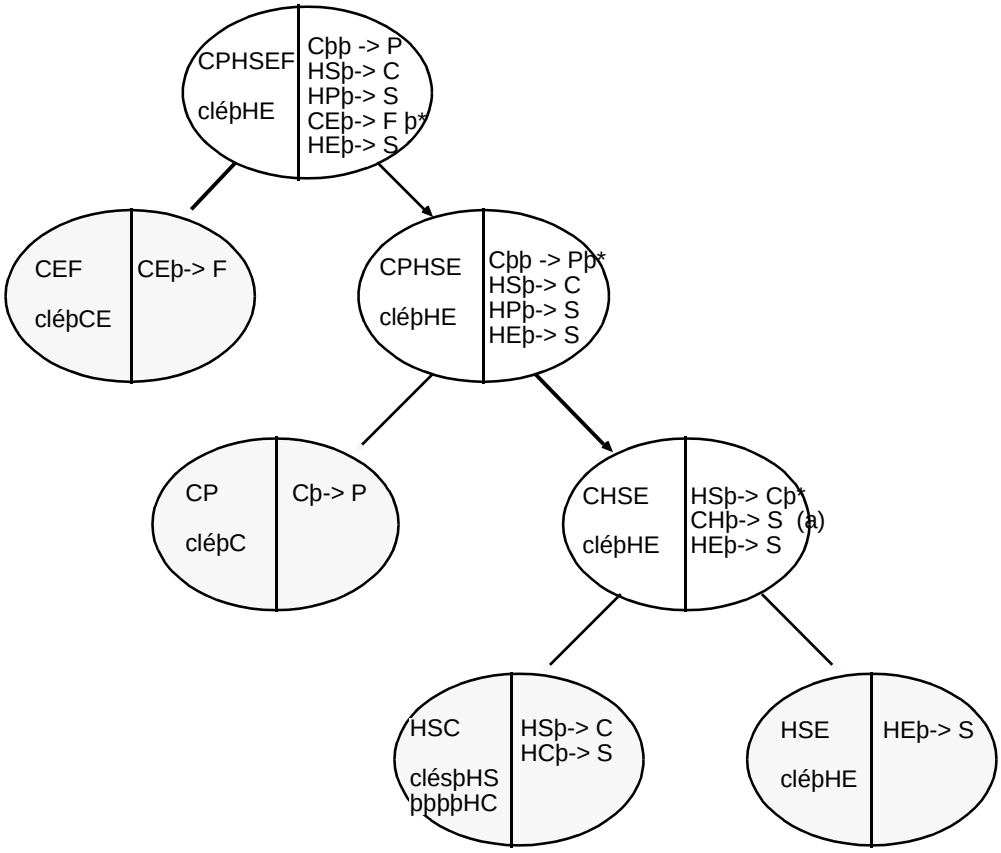
valeur finale de X : la collection de sous-schémas

Algorithme

procédure BCNF (U : attributs, F : dépendances) ;
début
  si <U, F> en BCNF
    alors
      X := X ∪ {<U, F>}
    sinon
      soit $X \rightarrow A \in F^+$, X ne contenant pas de clé ;
      BCNF (XA, F[XA]) ;
      BCNF (U - A, F[U - A])
  fsi
fin ;
X := ∅ ;
BCNF(U_0, F_0) ;

Exemple
U_0 = {Cours, Prof, Heure, Salle, Etudiant, Formation}
F_0 = {C → P, HS → C, HP → S, CE → F, HE → S}

- La seule clé est HE. A chaque pas de l'algorithme on note par un * la dépendance choisie.



- On remarque que, lors de la seconde étape, on a perdu la dépendance HP $\rightarrow$ S.
- On remarque également que la dépendance notée (a) n'apparaissait pas explicitement au pas précédent: on rappelle que c'est bien F^+ qui doit être projeté.
- Enfin, il est clair qu'en choisissant d'autres dépendances que celles qui ont été retenues on obtiendrait un résultat différent.

4.2.5.2 Décomposition en 3NF

Principe

À partir d'un schéma <U, F> dont K est une clé:

1. On cherche une couverture minimale G pour F,
2. Pour chaque partie gauche X des dépendances de G telle que $X \rightarrow A_1$, …, $X \rightarrow A_n$, on produit un sous-schéma $<XA_1 \ldots A_n, G[XA_1 \ldots A_n]>$,
3. Si aucun des sous-schémas produits ne contient de clé, on ajoute le sous-schéma <K, ∅>.

Algorithme de décomposition en 3NF

Entrée

<U, F>, schéma initial

Structure de données

S, ensemble de couples <U, F>

Sortie

valeur finale de S: la collection de sous-schémas

Algorithme

1) G := couverture_minimale(F) ;
2) S := ∅ ;
   pourtout X telque $X \rightarrow A_1$, ..., $X \rightarrow A_n$
   (i.e. soit les DF de G de partie gauche X)
   faire
   S := S ∪ {<$XA_1 \ldots A_n$, G[$XA_1 \ldots A_n$]>};
   fait ;
3) si aucun élément de S ne contient de clé
   alors
   K := une clé ;
   S := S ∪ {<K, {∅}>};
   fsi ;

Remarques

1. C'est le fait de partir d'une couverture minimale qui assure que les sous-schémas obtenus sont en 3NF.

2. La préservation des dépendances est, à partir de G, triviale.

3. C'est l'ajout d'une clé qui assure la validité de la décomposition.

4. Simplification : sans modifier les propriétés générales de l'algorithme, on peut simplifier le résultat S en retirant tout sous-schéma contenu dans un autre sous-schéma de S. Cette étape n'est pas nécessaire, mais elle simplifie le résultat en éliminant les redondances.

Exemples

• La base transport:<R (f, ville, frais), F : {f → ville, ville → frais, f → frais}>
1) G = {f → ville, ville → frais}
2) S = {<f ville, {f → ville}>, <ville frais, {ville → frais}>}
3) La clé f est utilisée dans S ; on n'ajoute rien

• U = {**C**ours, **P**rof, **H**eure, **S**alle, **E**tudiant, **F**ormation}
F = {C → P, HS → C, HP → S, CE → F, HE → S}
1) F est minimal.
2) On obtient δ = {CP, HSC, HPS, CEF, HES}.
3) Il n'y a rien à ajouter.

4.3 Conclusions

Nous effectuons, en guise de conclusion à ce chapitre, quelques remarques sur l'application des techniques que nous avons évoquées. Pour ce faire nous traitons un exemple simple.

Une société d'informatique veut implanter une base de données sur des logiciels (qu'elle utilise ou qu'elle pourrait utiliser), et sur les divers projets qu'elle conduit.

Une première phase de l'analyse vise à préciser:

- les informations qu'il faut enregistrer,
- les liens logiques entre ces diverses informations,
- les usages attendus du produit informatique (états, modifications et consultations interactives …) ainsi que ses limites.

Nous supposons que cette étude a permis de dégager les éléments qui suivent.

1. Pour les logiciels, on doit disposer d'informations sur:

- des fournisseurs de logiciel: nom et adresse,
- des logiciels: nom, type (langage, éditeur, tableur …), fournisseur et machine (PC, SUN …).

Pour un même logiciel, il peut exister plusieurs versions (au moins une) identifiées par un nom de version (V3.2 …), une date de disponibilité et un prix.

La société souhaite pouvoir archiver des logiciels dont elle ne se sert pas.

2. La société est structurée en projets qui ont un nom et un responsable. On doit pouvoir connaître les machines utilisées dans chaque projet. Par machine, on entend « un SUN » et non « le SUN situé sur la deuxième table du bureau 36 ».

3. Les projets utilisent certaines versions de logiciels. Un projet n'utilise cependant que des logiciels qui tournent sur une machine dont il dispose. Par contre il est possible qu'un projet dispose de machines pour lesquelles il n'utilise pas de logiciel décrit dans la base.

On s'intéresse à la suite de l'analyse.

Etape 1. Détermination des attributs et des dépendances fonctionnelles associées.
À ce stade, on essaie de repérer des entités significatives auxquelles on associe un attribut de codage. Les liens logiques entre les entités engendrent le plus souvent des dépendances fonctionnelles entre ces attributs de codage. On prend soin, toutefois, de bien repérer ce qui ne peut être capté à l'aide de dépendances fonctionnelles.

1) Entité fournisseur

Attributs:f (code), nom_f (nom), adresse
DF: f → nom_f adresse

2) Entité logiciel

Pour déterminer cette entité il y a quelques problèmes: *pascal*, fourni par *Borland* et *MicroSoft*, doit-il avoir un ou plusieurs numéros de code ? En fait, on sait qu'on achète un logiciel à un constructeur pour un type donné de machine. C'est cette entité globale qui engendre des versions, des prix … D'où le choix:

Attributs:l (code), nom_l (nom), type, m (machine),
v (version), date, prix
DF: l → nom_l type f m
l v → date prix

3) Entité projet

Attributs:p (code), nom_p (nom), r (responsable)
DF: p → nom_p r

Remarque: Le lecteur aura compris de lui-même que m, v et r sont des codages engendrant des relations (entités machines, versions et responsables) que nous ne détaillons pas.

Il importe de noter deux liens logiques et une contrainte qui ne se traduisent par aucune dépendance fonctionnelle:

- les projets disposent de machine,
- les projets utilisent des versions de logiciel,
- un projet n'utilise que des logiciels qui tournent sur une machine dont il dispose.

A la fin de cette première étape on retient donc :

U = f nom_f adresse l nom_l type m v date prix p nom_p r
F = { f → nom_f adresse,
l → nom_l type f m,
l v → date prix,
p → nom_p r}
dont la clé, unique, est p v l.

Etape 2. Mise en 3NF
F est une couverture minimale factorisée ; aucun élément ne contient de clé. On obtient donc:

fournisseur (f, nom_f, adresse)
logiciel (l, nom_l, type, f, m)
version (v, l, date, prix)
projet (p, nom_p, r)
clé (p, v, l)

Etape 3. Critique du système obtenu

On remarque d'abord que les relations sont en BCNF ce qui est encourageant.

Il faut ensuite se poser des questions sur:

- la relation clé, artificielle.
- les liens ou contraintes non traduites.

1) La relation clé.

- La relation clé a été ajoutée pour assurer la validité de la décomposition. Une autre manière de décrire le même phénomène consiste à dire que rien ne reliant projets, versions et logiciels il faut disposer d'une relation de jointure.
- Dans ce cas particulier, cependant, on remarque qu'il existe un lien entre ces entités, lequel lien n'est capté par aucune dépendance fonctionnelle: les projets utilisent des versions de logiciel.
- Il est donc tout a fait légitime d'introduire, en lieu et place de la relation clé, la bonne relation de jointure:

utilise (p, v, l)

2) Liens ou contraintes non traduites

- Le seul lien non encore traduit est: les projets disposent de machine.
- En effet, dans le schéma courant, on peut, par jointure, relier p et m: on n'obtient cependant, pour un projet, que les machines pour lesquelles il utilise une version d'un logiciel ; on « manque » donc les machines pour lesquelles le projet n'utilise pas de logiciel décrit dans la base. L'ajout d'une nouvelle relation s'impose:

dispose (p, m)

- Il reste à intégrer la contrainte: un projet n'utilise que des logiciels qui tournent sur une machine dont il dispose.
- Cette contrainte doit faire l'objet de contrôles explicites lors des opérations de mise à jour.

La dernière étape de cette analyse éclaire la citation donnée en tête de ce chapitre. Le lecteur est invité à utiliser les méthodes que nous avons données sans en attendre de miracles et surtout à ne pas laisser son bon sens au vestiaire.

CHAPITRE 5

Exploitation d'une base de données

Nous faisons, dans ce chapitre, le point sur les principaux problèmes liés à l'exploitation d'une base de données.

5.1 La concurrence d'accès

Le contrôle des transactions et la concurrence d'accès ont déjà été évoqués au paragraphe 3.4.2. Nous voulons apporter ici quelques compléments.

5.1.1 Deux problèmes à résoudre

5.1.1.1 Perte de mise à jour

Soit le programme diminuer (v) qui consiste à décrémenter d'une valeur v, dans Mafourniture, la quantité du produit p5 fourni par f1.

Un te1 programme correspond à l'exécution des trois instructions suivantes:

a - lire Mafourniture (f1, p5)
b - qte <-- qte - v
c - écrire Mafourniture (f1, p5)

Supposons que 2 transactions T1 et T2 utilisent le programme diminuer:

T1: diminuer (2) {actions 1a, 1b[2], 1c}
T2: diminuer (5) {actions 2a, 2b[5], 2c}

Si les actions de T1 et de T2 sont bien synchronisées, la quantité fournie peut, par exemple, passer de 8 à 1. En revanche, si les actions de T1 et de T2 se déroulent selon l'ordre: 1a, 1b, 2a, 2b, 1c, 2c, la mise à jour effectuée par T1 est écrasée et la quantité devient 3. La mise à jour de T1 est perdue.

5.1.1.2 Lecture impropre

Premier exemple
Soit la table compte (numéro_compte, solde).

Une transaction T1 enregistre un virement interne d'un compte à un autre par:

1. Débit d'un premier compte
2. Crédit d'un deuxième compte.

La transaction T2 exécute l'instruction:

```
select sum(solde) from compte
```

Si cette instruction est exécutée entre l'enregistrement du débit et celui du crédit et qu'elle a connaissance des données modifiées par le débit, la lecture est dite impropre.

Deuxième exemple

1. T1 effectue des mises à jour sans les valider,
2. T2 lit des données mises à jour par T1 et a connaissance de ces mises à jour,
3. T1 annule ensuite ses mises à jour.

La lecture de T1 était impropre.

5.1.1.3 Notion de transaction

Une transaction a été définie au chapitre 3 comme une unité atomique de traitement, constituant un tout au regard des pannes éventuelles et des accès multiples.

Une transaction comportant de la mise à jour doit avoir les quatre propriétés connues sous l'appellation ACID :

- Atomicité: une transaction est exécutée totalement ou pas du tout;
- Correction ou Cohérence: en fin de transaction, la cohérence est respectée ;
- Isolation: aucune mise à jour n'est visible de l'extérieur de la transaction avant qu'elle ne soit finie et validée ; il y a isolation de la transaction;
- Durabilité: les mises à jour sont conservées durablement.

Imposer aux transactions le respect des conditions ACID permet de résoudre les problèmes que nous venons de voir. Il faut noter que nous allons présenter une définition plus contraignante de l'isolation un peu plus loin.

5.1.2 Lecture non reproductible

Deux transactions T1 et T2 s'exécutent en parallèle. La transaction T1 effectue deux fois la même instruction:

```
select qte
from mafourniture
where f = 'f1' and p = 'p5'
```

Entre les deux exécutions de cette instruction, la transaction T2 appelle le programme diminuer déjà évoqué, pour modifier la quantité qui nous intéresse et cette transaction valide la modification par commit. Pour T1, les deux exécutions de la même instruction ne donnent pas le même résultat.

C'est le phénomène de lecture non reproductible, qui peut être gênant, par exemple lors de la consultation de gros volumes de données, avec des recoupements à effectuer entre les données nécessitant plusieurs lectures des mêmes informations. Ceci correspond typiquement à un traitement statistique.

Pour le SQL d'Oracle, l'instruction:

```
set transaction read only
```

assure cette lecture reproductible; l'utilisateur ne peut faire de mise à jour, mais ne voit pas les mises à jour éventuelles faites par les autres, même si elles ont été validées par commit.

Le niveau d'isolation atteint pour une lecture reproductible est plus fort que ce qui a été présenté avec les propriétés ACID d'une transaction. Ce niveau d'isolation est dit *sérialisable*: l'exécution concurrente de transactions produit le même effet qu'une exécution séquentielle. L'isolation sérialisable:

- suppose la lecture répétitive (le n-uplet relu n'a pas été modifié ou, s'il a été modifié, le changement n'est pas obtenu)
- interdit l'existence de n-uplet dit fantôme, dont un exemple correspond à la séquence d'opérations suivantes:

  1 -lecture de tous les fournisseurs localisés à Paris

  2 -insertion d'un nouveau fournisseur de Paris par une autre transaction et validation

  3 -relecture des fournisseurs de Paris avec un n-uplet fantôme, absent en 1.

5.1.3 Les verrous explicites

Nous avons vu que les instructions de mise à jour posent implicitement des verrous, sans qu'il soit nécessaire de programmer cette pose. De même, les instructions du DDL (create table, alter table...) posent des verrous implicites.

La norme SQL2 précise que, si un SGBD offre un niveau d'isolation moins fort que celui qui correspond à l'isolation sérialisable, il doit fournir des instructions permettant la pose explicite de verrous. Cette pose peut se faire:

- au niveau table, par l'instruction lock table: verrouillage d'une table dans un mode qui admet plus ou moins de partage avec les autres,
- au niveau n-uplet: verrouillage exclusif d'un ou de plusieurs n-uplets.

Ces verrouillages explicites sont utiles pour réserver des données avant des mises à jour assez importantes. Ils sont parfois nécessaires.

Par exemple, en Oracle, la pose explicite de verrous permet d'assurer une lecture reproductible à une transaction qui doit effectuer de la mise à jour.

Les verrous, implicites ou explicites, doivent toujours rester posés un minimum de temps. Pour cela, il est conseillé de:

- préparer les mises à jour (saisir avant de verrouiller),
- accélérer les traitements pendant le verrouillage (cf. fin de ce chapitre),
- appliquer l'instruction commit dès que possible.

Si on s'impose de ne pas utiliser un objet sans l'avoir verrouillé, on peut remarquer que l'emploi d'un protocole de verrouillage dit *à deux phases* (on attend d'avoir posé l'ensemble des verrous avant d'effectuer le premier déverrouillage) assure l'isolation sérialisable.

5.1.4 Résolution de l'interblocage

Voici un exemple d'interblocage (ou verrou mortel) entre deux transactions:

- T1 a posé un verrou exclusif sur des données D1 (une table ou un ou plusieurs n-uplets),
- T2 a posé un verrou exclusif sur des données D2,
- T1 veut poser un verrou exclusif sur D2; il est mis en attente,
- T2 veut poser un verrou exclusif sur D1; une situation d'interblocage est détectée, c'est-à-dire l'existence de transactions en attente de ressources verrouillées. T1 attend T2 et T2 attend T1.

Deux mécanismes existent pour résoudre le problème d'interblocage:

- *Détection et résolution*: il y a détection d'un cycle dans le graphe d'attente et résolution par le SGBD qui choisit une transaction victime qui doit annuler ses actions par rollback.
- *Prévention*: des contraintes fortes sont imposées pour éviter que le problème ne se pose. Ce type de solution est retenu dans les bases de données réparties.

D'une manière générale, si l'on est capable d'imposer aux transactions leur verrouillage selon le même ordre, on élimine tout risque d'interblocage.

5.2 L'interface entre un SGBD et un langage de programmation

5.2.1 Présentation générale

Il y a trois types d'approche pour réaliser cette interface:

- L'appel de primitives traitant la base de données: depuis le programme en langage-hôte, des primitives sont appelées: procédures externes, sous-programmes ou classes selon les types de langage. L'avantage de cette solution est qu'il n'y a rien à changer dans le langage-hôte lui-même. Les interfaces de bas niveau (mais pas seulement elles) sont de ce type.
- L'extension d'un langage de programmation par un langage de manipulation de données consiste à considérer un langage et à proposer des extensions qui font partie du nouveau langage; par exemple, Cobol étendu peut devenir cobol/IDS II (SGBD réseau), C étendu peut devenir O_2C (SGBD objet).
- L'intégration d'instructions propres à la base de données dans le programme en langage hôte. Ces instructions sont reconnues, car elles commencent par une indication spéciale. Ainsi, SQL2 fait commencer ces instructions par:
  exec sql.
  Cette version imbriquée dans un langage hôte s'appelle *embedded SQL* et est présentée au paragraphe suivant.

  Deux types de mise en oeuvre sont possibles: une pré-compilation des requêtes traitant la base de données, qui traduit ces requêtes en du code en langage hôte, destiné au compilateur habituel du langage hôte; une interprétation de ces requêtes à l'exécution, solution plus simple à mettre en oeuvre, mais plus coûteuse. C'est le premier type de mise en oeuvre qui est usuel. Voici un schéma récapitulatif de la mise en oeuvre avec précompilation.

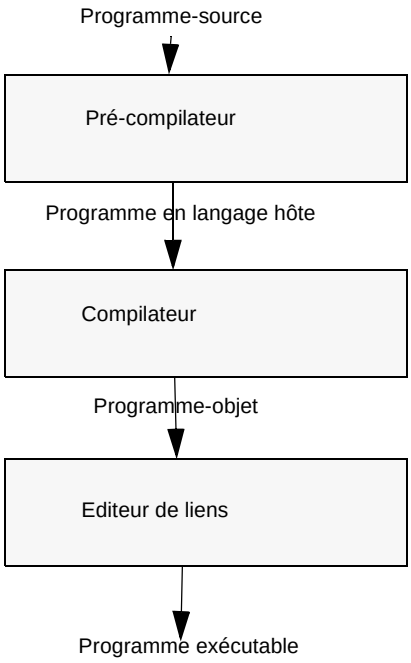


Figure5.1

5.2.2 Embedded SQL

Ce paragraphe est conforme à la norme SQL2.
Les instructions SQL doivent être préfixées par exec sql. Un langage hôte, dit de troisième génération, à la différence de SQL, n'offre pas le concept d'ensemble; une requête SQL génère une table dans une zone de travail, dont chaque élément doit être traité tour à tour par le langage hôte. Pour traiter un par un chaque élément d'un ensemble généré par une requête SQL, le langage hôte doit donc utiliser un pointeur appelé curseur (cursor). Le curseur doit être déclaré et associé à une requête SQL, puis ouvert (open) et fermé (close).

SQL2 permet de:

- lire un élément pointé par le curseur (fetch)
- le mettre à jour ou le supprimer (delete/update ... where current...)

Voici un exemple concernant un traitement sur tous les codes de fournisseur de la ville de Paris.

```
exec sql declare c1 cursor for
select f from fournisseur
where ville = 'Paris'
```

```
order by f ;
exec sql begin declare section ;
        varchar code [5] ;
exec sql end declare section ;
exec sql open c1 ;
<pour toutes les valeurs retenues du code>
exec sql fetch c1 into :code ;
<traitement sur code>
exec sql close c1 ;
```

La variable SQL de nom standard sqlcode vaut 0 tant qu'il reste au moins un fournisseur à traiter et qu'aucune erreur ne se produit, 100 après le traitement du dernier fournisseur et a une valeur négative en cas d'erreur.

5.3 Le développement d'applications

Un SGBD doit permettre la conception et la réalisation d'applications, à des fins d'interrogation et de mise à jour d'une base de données.

Ces applications doivent satisfaire divers objectifs, notamment:

- être bien pensées, c'est-à-dire offrir toute garantie lors de mises à jour: vérifications, contrôles de cohérence,
- être « fignolées » du point de vue de l'édition des résultats, c'est-à-dire avoir des écrans et des états bien présentés,
- être d'utilisation souple, car bien souvent destinées à des utilisateurs finaux non informaticiens, d'où par exemple, menus déroulants, dialogues avec l'utilisateur par question/réponse.

Plusieurs possibilités sont offertes pour le développement d'applications:

1. Utiliser le langage de manipulation des données de SQL, c'est-à-dire construire un fichier de requêtes SQL.
   Il y a beaucoup de limitations par rapport aux objectifs, car on ne dispose pas de variables à usage général, ni des structures de contrôle habituelles d'un langage procédural, ni d'E/S (pour générer des écrans ou bâtir des dialogues).
2. Utiliser un langage procédural spécialement conçu (PL/SQL pour Oracle) offrant les variables, les structures de contrôle habituelles (conditionnelles, boucles en plus du langage SQL).

   Cette possibilité est avantageuse car la programmation est aisée mais il y a encore des limitations: le langage n'offre pas toutes les possibilités d'un langage de programmation habituel (en particulier pas d'E/S).
3. Utiliser un langage de programmation standard (par exemple C).

   Remarquons que si l'on dispose de toute la puissance souhaitable, la programmation des E/S (menus, écrans, dialogues) est assez pénible !
4. Utiliser un outil logiciel spécialisé de développement (le *générateur d'application* Oracle*FORMS" par exemple).

De tels outils évitent au programmeur d'application d'avoir à recourir à une programmation classique, en lui offrant une convivialité (à base de menus, fenêtres) pour construire l'application et des facilités pour produire des écrans (éditeur de texte et de dessins).

Il s'agit d'une manière de programmer qui s'apparente à la programmation objet, où une application est composée de blocs (ou parties d'écran), eux-mêmes composés de champs, où des actions (requêtes SQL par exemple) associées à ces trois niveaux sont exécutées à l'occurrence d'événements auxquels sont attachées les actions.

La tendance générale est au développement des outils du type (4), souvent qualifiés de *langages de quatrième génération*.

5.4 L'administration de données

Ainsi que cela apparaît sur la figure 1.2 du chapitre 1, l'administrateur de base de données (*dba*, comme *database administrator*) a un double rôle: un rôle organisationnel et un rôle technique. Le rôle organisationnel concerne la responsabilité de la définition du modèle conceptuel des données et du partage de ces données par les applications. Quant au rôle technique, il consiste à mettre en oeuvre ce modèle et ce partage, au niveau physique, à l'aide des capacités techniques du SGBD. Les deux rôles peuvent être assurés par une ou plusieurs personnes.

Dans ce chapitre, nous nous intéressons au rôle technique de l'administrateur de données. Ce rôle consiste à effectuer les travaux suivants:

- Installation du SGBD et des outils associés: la première tâche de l'administrateur est d'installer le SGBD, ainsi que les outils qui lui sont associés.
- Création de la base de données et responsabilité de l'évolution: l'administrateur a la tâche de créer la base de données conformément aux modèles conceptuel et interne. Il doit assurer l'évolution de cette base en créant, en modifiant ou en supprimant certaines structures.
- Gestion des utilisateurs et des privilèges d'accès: l'administrateur crée ou supprime les utilisateurs, et attribue ou retire les privilèges d'accès aux données pour les différents utilisateurs.
- Assurer (ou superviser) les échanges de données entre la base de données et le monde extérieur: l'administeur doit réaliser l'intégration des données en provenance d'autres applications ou bases de données, et faire migrer les données de la base vers d'autres applications ou bases de données.
- Amélioration des performances: l'administrateur doit choisir l'implantation optimale des données de façon à obtenir les meilleures performances. Il doit connaître l'architecture interne du logiciel, ainsi que tout ce qui permet d'accélérer le traitement des requêtes.

- Gestion de la sécurité et de la cohérence des données: l'administrateur doit mettre en place les structures et les procédures, permettant de faire face à tous les incidents, et de retrouver l'intégrité et la cohérence des données.

5.5 Le dictionnaire de données

Les SGBD maintiennent dans un dictionnaire un ensemble d'informations sur les usagers et leurs droits, sur les tables, les vues, les index ... Des recommandations SQL concernent ce dictionnaire de données. Chaque fois qu'une table ou un objet de la base est ajouté, modifié ou supprimé, le dictionnaire doit automatiquement être mis à jour. Le dictionnaire est lui-même constitué de plusieurs tables (ou de vues), que l'on peut interroger et manipuler grâce à SQL.

Il existe trois niveaux dans les tables ou vues du dictionnaire; le niveau est précisé par le préfixe du nom de la table ou de la vue:

dba_	*tables réservées au(x) administrateur(s); un administrateur peut accéder à tous les objets et à toutes les données de la base entière;*
all_	tables *concernant tout ce qui est accessible à l'utilisateur;*
user_	tables *concernant tout ce qui appartient à l'utilisateur.*

Chacun peut utiliser les tables des deux derniers niveaux.

Exemples

```
select *
from dba_users
```
fournit la liste des usagers pour un administrateur

```
select *
from all_tables
```
fournit l'ensemble des tables accessibles par l'utilisateur

```
select *
from user_objects
```
fournit la liste des objets (tables, vues…) possédés par l'utilisateur.

5.6 Le recouvrement de transactions

5.6.1 Les trois types de reprise

Le *recouvrement* de transactions consiste à retrouver un état cohérent en revenant à une situation antérieure qui prend en compte, si c'est possible, les mises à jour faites par toute transaction validée. Trois types de reprise sont possibles: la *reprise annulation*, la *reprise à chaud*, la *reprise à froid*.

- La reprise annulation : on peut annuler une transaction suite à une erreur ou à un interblocage. Il faut alors « défaire » une ou plusieurs transac-

tions pour revenir à la situation cohérente correspondant au début de la ou des transactions interrompues.

- La reprise à chaud : en cas de panne incontrôlée où le stockage volatil (en mémoire centrale) est perdu, il faut repartir du dernier *point de reprise*. Un point de reprise est un point dans le temps où les transactions ont vu tous leurs résultats écrits en mémoire secondaire permanente (dans la base de données ou dans les fichiers journaux sur disque). Depuis le dernier point de reprise, il faut refaire les transactions ayant effectué leur validation avant la panne et défaire les actions en cours non terminées lors de la panne.
- La reprise à froid : en cas de panne de la mémoire secondaire (perte du stockage fiable et permanent), il faut repartir d'une sauvegarde des données et refaire toutes les transactions ayant effectué leur validation avant la panne.

  Les reprises à froid supposent donc des archivages. Typiquement, une sauvegarde de la base entière peut se faire en fin de mois; une sauvegarde cumulative peut se faire en fin de semaine et contient les mises à jour depuis la dernière sauvegarde cumulative (ou depuis la dernière sauvegarde de base entière, pour la première semaine du mois) ; une sauvegarde incrémentale peut se faire en fin de journée et contient les mises à jour depuis la dernière sauvegarde incrémentale (ou depuis la dernière sauvegarde cumulative, pour le premier jour de la semaine). Ainsi, sans sauvegarder toute la base chaque soir, on peut en cas de besoin retrouver la situation du soir précédent. La périodicité des différentes sauvegardes sera évidemment adaptée au contexte de l'entreprise.

5.6.2 Illustration des trois types de reprise

La figure suivante illustre les trois types de recouvrement en cas de problème. Dans cette figure, P correspond au problème (panne de site ou problème moins important); R correspond au dernier point de reprise; S correspond à la dernière sauvegarde. T1, T2, T3, T4 et T5 sont des transactions.

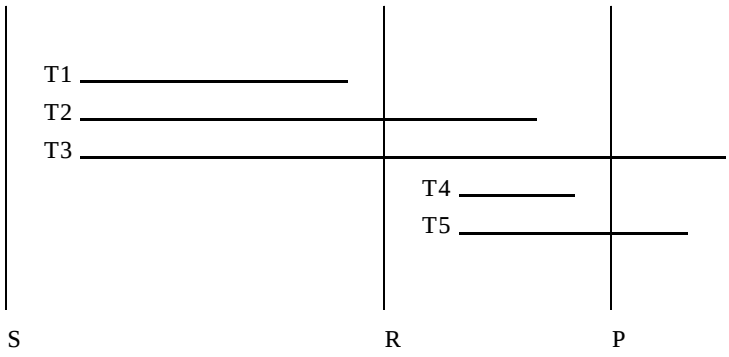


Figure5.2

Les trois types de reprise sont les suivants:

- reprise annulation: en partant de P, défaire T3 et T5,
- reprise à chaud: en partant de R, défaire les actions de T3 avant R, puis refaire les actions de T2 après R et de T4,
- reprise à froid: en partant de S, refaire T1, T2 et T4, si c'est possible.

5.6.3 Informations à conserver dans les journaux

L'action de défaire ne peut s'appliquer qu'à une transaction n'ayant pas encore effectué sa validation; les informations qu'elle nécessite n'ont pas besoin d'être conservées après la validation de la transaction. Ces informations consistent dans les valeurs des données avant mise à jour, dites images-avant.

Au contraire, l'action de refaire peut être déclenchée à tout moment. Les informations qu'elle nécessite sont les nouvelles valeurs des données, dites images-après.

Les cas de reprise les plus fréquents correspondent à la reprise annulation. Aussi, les images-avant utilisées seules pour ce type de reprise peuvent être conservées dans la base elle-même; on peut parler alors de *données d'annulation*. Les journaux de reprise sont des fichiers indépendants de la base de données, qui peuvent contenir les images-avant et les images-après.

Ce sont les valeurs avant et après mise à jour qui sont les plus simples à conserver dans les journaux. Il y a des techniques plus sophistiquées de journaux différentiels qui stockent les différences entre les anciennes et les nouvelles données, ou de journaux d'opérations qui stockent des informations sur les opérations.

5.7 Une architecture fonctionnelle de référence

L'architecture à trois niveaux vue au chapitre 1 est souvent remplacée par une architecture qui intègre le niveau conceptuel et le niveau interne dans un seul niveau. En fait, le niveau conceptuel est souvent pris en charge par des outils d'aide à la conception extérieurs au SGBD. Restent alors deux niveaux: le schéma (conceptuel et interne) et les vues.

Examinons les quatre étapes subies par une requête d'interrogation ou de mise à jour.

- La première étape est réalisée par un analyseur. Elle consiste dans l'analyse syntaxique (conformité à la grammaire) et sémantique (conformité à la

vue référencée et au schéma) de la requête. La requête est alors traduite en format interne.

- La deuxième étape est réalisée par un traducteur. En effet, une requête qui référence une vue doit être traduite en une ou plusieurs requêtes référençant des objets existant dans la base, qui eux figurent au niveau du schéma. C'est aussi au niveau du traducteur que sont pris en compte les problèmes de contrôle de droits d'accès et prévus les contrôles d'intégrité lors des mises à jour.
- La troisième étape est confiée à un optimiseur, dont le rôle essentiel est d'élaborer un plan d'exécution optimisé pour traiter une requête. L'optimiseur décompose une requête en opérations d'accès élémentaires (telles que: lecture d'article, sélection d'index) et choisit l'ordre qui lui semble le meilleur pour réaliser ces opérations. L'optimiseur tient compte des accélérateurs d'accès utilisables, des différentes manières de faire une opération telle qu'une jointure… Il s'appuie souvent sur un modèle de coût qui permet d'évaluer le coût d'un plan d'accès avant son exécution, en fonction d'un estimé de la taille des données initiales et intermédiaires. Des statistiques sur les données initiales sont alors utilisées.
- La quatrième étape est réalisée par un exécuteur, chargé d'exécuter le plan d'accès résultant de l'étape 3. C'est à ce niveau que sont réglés les problèmes de concurrence d'accès.

Le schéma qui suit récapitule les quatre étapes.

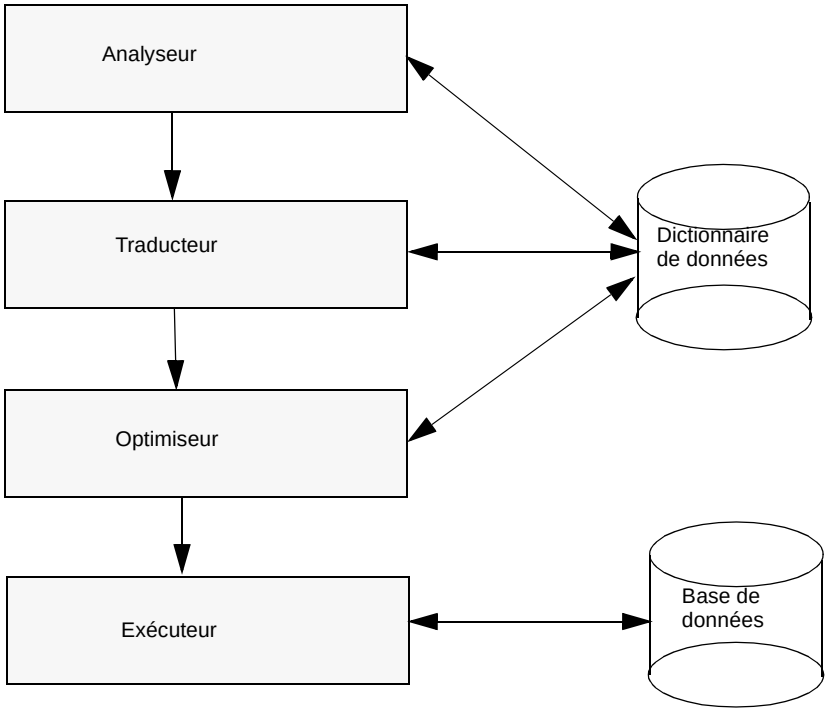


Figure 5.3 Traitement d'une requête

5.8 L'évaluation de requêtes

Etudions plus en détail la manière avec laquelle un optimiseur réalise un plan d'exécution. Nous avons déjà vu quelques éléments de mise en oeuvre à la fin du chapitre sur l'algèbre relationnelle, à savoir que le SGBD peut utiliser des algorithmes performants de jointure, qu'il peut utiliser des chemins d'accès et qu'il peut transformer les requêtes. Il ressort de ces éléments que l'optimiseur doit éviter une jointure par boucles imbriquées (produit cartésien), qu'il doit appliquer une sélection le plus tôt possible, qu'il doit utiliser un index qui diminue le nombre d'entrées-sorties, sachant que ces entrées-sorties jouent un rôle prépondérant dans l'optimisation.

Nous distinguons deux méthodes d'optimisation: l'une statique, l'autre statistique.

5.8.1 Méthode statique

L'optimiseur choisit le plan d'exécution en fonction de règles d'optimisation utilisant notamment les chemins d'accès disponibles pour chaque table qui intervient dans la requête SQL. Ainsi, les méthodes d'accès sont classées en ordre décroissant d'intérêt, depuis la meilleure (accès à un ou des n-uplets de numéro interne ou adresse connue) jusqu'à la plus simple (accès séquentiel). Entre les deux, peuvent figurer les accès utilisant:

- la notion de regroupement ou jointures anticipées, c'est-à-dire de voisinage physique sur disque pour des n-uplets souvent utilisés ensemble, tels que un fournisseur et ses fournitures, ou une commande et ses lignes de commande,
- une technique de hachage,
- un index, avec différents niveaux de qualité tels que: clé primaire ou pas, attribut simple ou attributs composés, bits de poids fort d'une clé, intervalle.

Exemple
Soit la requête SQL sur la base épicerie:

```
select f from fournisseur
where ville  = 'Paris' and remise > 8
```

Nous supposons qu'un index primaire est associé à l'attribut f et que deux index secondaires sont associés l'un à ville, l'autre à remise. Il y a trois chemins disponibles:

- lecture séquentielle de la table fournisseur,
- accès par l'index sur la remise,
- accès par l'index sur la ville.

Avec la méthode statique, c'est l'accès par l'index sur la ville qui est utilisé, car l'opérateur = l'emporte sur l'opérateur >.

5.8.2 Méthode statistique

Cette méthode se base à la fois sur les principes vus précédemment et sur des statistiques relatives aux tables concernées par une requête, pour déterminer le plan d'exécution de moindre coût estimé.

Ces statistiques peuvent porter notamment sur le volume d'une table (nombre de n-uplets et longueur moyenne d'un n-uplet), sur le nombre de blocs alloués à la table en mémoire secondaire, sur le nombre de valeurs distinctes par attribut, sur les valeurs minimales et maximales de chaque attribut. Elles sont utilisées pour déterminer soit le meilleur temps de réponse pour obtenir tous les n-uplets d'une requête, soit le meilleur temps de réponse pour extraire les premiers n-uplets du résultat.

Le comportement de l'optimiseur pour le choix de la méthode et du but d'optimisation (tous les n-uplets ou les premiers n-uplets) peut être défini au niveau de la base de données, au niveau d'une session SQL ou lors d'une instruction, dans laquelle le programmeur peut forcer l'optimiseur à utiliser un chemin spécifique.

Exemple
Reprenons le même exemple que pour la méthode statique.

Si le nombre de villes différentes est petit, par exemple 4, et que le plus grand taux de remise vaut 10, le critère de sélection sur la ville peut être jugé comme moins discriminant que celui sur le taux de remise. Dans ce cas, l'accès se fera en utilisant l'index sur le taux de remise.

La lecture séquentielle d'une table par un SGBD est souvent réalisée à l'économie grâce à une lecture de plusieurs blocs en une seule opération d'entrée-sortie physique. Dans un tel contexte, selon les cas, un accès séquentiel peut s'avérer satisfaisant. Ainsi, lorsqu'un grand nombre des n-uplets d'une table est concerné par une requête, un accès séquentiel est plus performant qu'un accès indexé.

Dans l'exemple qui nous intéresse ici, la lecture séquentielle de la table fournisseur peut donc être préférable si aucun des critères de sélection n'est jugé assez discriminant.

5.8.3 Conclusion sur l'évaluation de requêtes

L'optimisation basée sur les statistiques donne en général les meilleurs résultats. L'optimisation statique est à retenir lorsque la création des statistiques n'est pas possible, ou lorsque la taille des tables et la distribution des données changent souvent, ou lorsqu'une optimisation manuelle des jointures a pu être faite.

Enfin, il faut noter que l'optimisation globale des temps de réponse tient compte aussi de facteurs comme:

- la taille disponible en mémoire centrale; ainsi, une taille importante favorise certaines jointures,
- la gestion des tampons et des caches,
- le placement des n-uplets dans les blocs,
- le verrouillage des n-uplets pour les accès concurrents,
- la gestion des tampons associés aux journaux de reprise et aux données d'annulation.

CHAPITRE 6

Informatique et législation

6.1 Avertissement

Ce chapitre n'est pas à jour: les textes cités ont, en général, été modifiés depuis la rédaction. Cependant, l'esprit des lois citées est, pour l'essentiel, conservé. Pour obtenir une information plus récente, le lecteur pourra consulter divers sites *Web*, comme:

- http://www.sg.cnrs.fr/internet/legislation.htm
- http://www.cru.fr/securite/1INDEXs/Lois.html

6.2 Introduction

L'informatique est une technique relativement neuve ; pourtant, son utilisation, du moins dans les pays industrialisés, croît d'une manière que certains peuvent trouver inquiétante. Cette croissance un peu « sauvage » n'est pas sans poser des problèmes de société : craintes pour les libertés, craintes de chômage, craintes de déqualification, risques économiques … Ces interrogations légitimes ont donné lieu, dans la plupart des pays, à des législations particulières. Pour ce qui concerne la France, il y a, à l'heure actuelle, divers textes législatifs qui portent explicitement sur l'informatique.

- la loi n° 78-17 du 6 janvier 1978, intitulée « *Informatique, fichiers et libertés* »
- la loi n° 85-660 du 3 juillet 1985, relative aux « *droits d'auteur et aux droits des artistes-interprètes, des producteurs de phonogrammes et de vidéogrammes et des entreprises de communication audiovisuelle* » qui comporte un titre V intitulé « *Des logiciels* ».
- la loi n° 88-19 du 5 janvier 1988, dite « loi Godfrain », relative à la fraude informatique.
- la loi n° 90-1170 du 29 décembre 1990, relative à la cryptologie.

À ces textes il convient d'ajouter la jurisprudence, les délibérations de la CNIL, divers notes et décrets, la réglementation européenne… ainsi que les éventuels oublis des auteurs, qui n'ayant aucune qualification juridique, ne prétendent en aucun cas à l'exhaustivité.

Ces textes existent et tout informaticien doit, dans sa pratique professionnelle, les respecter. Comme le premier des textes cités touche directement au domaine visé par ce cours, il nous a semblé utile de les décrire très sommairement.

6.3 La loi « Informatique, fichiers et libertés »

La loi du 6 janvier 1978 concerne les traitements informatiques d'informations nominatives. Elle vise à assurer aux citoyens un certain nombre de garanties ; pour ce faire elle impose des obligations aux organismes opérant de tels traitements et, en conséquence, elle instaure un outil de contrôle, la Commission Nationale de l'Informatique et des Libertés (CNIL).

6.3.1 Traitements informatiques d'informations nominatives

Les informations dites nominatives sont celles « *qui permettent, sous quelque forme que ce soit, directement ou non, l'identification des personnes physiques* ».

On remarque que la possibilité d'indirection et l'imprécision sur la forme étendent, d'une certaine manière, le champ de la loi à des fichiers « manuels » susceptibles d'être utilisés en complément de fichiers informatiques. La loi vise à réglementer la collecte, l'enregistrement et les éventuels traitements de telles informations.

6.3.2 Garanties offertes au citoyen

Les principales garanties offertes au citoyen sont celles du droit d'accès et du droit de rectification : il doit donc pouvoir connaître l'existence des fichiers où il est décrit (il n'y a pas de fichiers secrets), le contenu des informations le concernant afin d'imposer, en cas d'erreurs, des éventuelles modifications.

On note en revanche qu'il est plus difficile de s'opposer à l'enregistrement d'informations à son sujet : la loi prévoit toutefois la possibilité de s'opposer, pour des raisons légitimes, à ce que des informations nominatives fassent l'objet d'un traitement.

Il est cependant interdit d'enregistrer des données nominatives sur « *les origines raciales ou les opinions politiques, philosophiques ou religieuses ou les appartenances syndicales des personnes* ».

Enfin, l'usage du droit d'accès est particulier pour les traitements intéressant la sûreté de l'Etat, la défense et la sécurité publique (il s'exerce par un

intermédiaire et le requérant ne reçoit que la notification du fait que les vérifications ont été effectuées) ainsi que pour les informations à caractère médical (le requérant désigne un médecin pour le représenter).

6.3.3 Les obligations imposées aux organismes de traitement

Comme il ne peut pas exister de fichiers nominatifs secrets, tout fichier de ce type doit faire l'objet d'une déclaration, préalablement à sa mise en oeuvre ; dans cette déclaration, la finalité des traitements doit être précisée, ce qui vise à exclure tout détournement ultérieur.

En ce qui concerne la réaction à la déclaration, la loi distingue le cas des organismes publics (plus contraints) du cas des organismes privés : dans le cas des organismes publics, la déclaration est également une demande d'autorisation qui donne lieu à un avis motivé.

Les sanctions prévues en cas de non respect de la loi peuvent être lourdes : 6 mois à 3 ans de prison, 2 000 à 20 000 F d'amende pour un « oubli de déclaration » ; jusqu'à 5 ans de prison et 2 000 000 F d'amende pour une infraction sur la collecte ou la nature des informations enregistrées.

6.3.4 Moyens de contrôle

La loi instaure une commission, la CNIL, qui est une autorité administrative (et non judiciaire) dont les membres sont toutefois irrévocables pendant la durée de leur mandat.

Elle veille au respect de cette loi : en particulier, elle recueille les déclarations, elle assure la mise en oeuvre des droits d'accès et de rectification. Pour ce faire, elle est habilitée à effectuer des enquêtes.

Il semble clair que cet organisme a, en plus du travail qui consiste à faire respecter cette loi, une fonction de conseil et de prospective sur les problèmes de société posés par l'informatique.

6.3.5 Quelques commentaires

L'existence de cette loi est évidemment un progrès important, même si, à l'évidence, il s'agit d'un premier pas. Pour l'avenir et les évolutions possibles, il convient de noter deux points importants :

- il est sain, vu la rapidité et les caractères souvent contradictoires des évolutions de l'informatique, que la législation, elle, évolue plus lentement. À titre d'exemple, il est clair que les conséquences de l'usage de la micro-informatique ne sont pas intégrées dans cette loi conçue antérieurement à l'explosion du phénomène. Il nous semble que les conséquences sociales de ce développement ne sont pas encore toutes évidentes et une législation prématurée peut être lourde de conséquences pour l'avenir.
- les réseaux de transmission de données couvrent maintenant le monde entier. Ce point semble condamner toute législation purement nationale :

il serait désastreux de voir se constituer des « paradis de données » à l'instar des « paradis fiscaux » existants.

On conçoit souvent la protection des libertés du citoyen en termes de restrictions : limitations d'accès, d'enregistrement, de traitement. On oublie facilement que la liberté passe également par la garantie de possibilités d'usage de l'informatique. On peut donner deux exemples prévus par la loi :

- « *les églises et les groupements à caractère religieux, philosophique, politique ou syndical peuvent tenir registre de leurs membres ou de leur correspondants sous forme automatisée. Aucun contrôle ne peut être exercé, de ce chef, à leur encontre* ». Il est clair que cette dérogation explicite à la loi générale donne en fait au citoyen des moyens de se défendre.
- « *l'accès du fichier électoral est ouvert dans des conditions identiques aux candidats et aux partis politiques …* ». On imagine facilement les conséquences néfastes de restrictions d'accès à de tels fichiers.

On a facilement tendance, et cette tendance est généralement accentuée par la jeunesse, à privilégier le fond, ce qui est évidemment sain, mais à négliger la forme. À titre d'exemple, il est fréquent qu'un informaticien membre d'une association à caractère culturel, social ou caritatif apporte son savoir et ses compétences pour améliorer le fonctionnement de l'association. Il est malheureusement fréquent que, dans un tel cas, il considère que les fichiers qu'il constitue sont de « bons » fichiers ne présentant aucun danger pour les libertés. Cette disposition d'esprit est source de négligences (non respect de la loi par exemple) qui ne seraient sans doute pas commises dans un contexte professionnel. Je prétends que cette attitude, profondément humaine, est dangereuse : aucune donnée n'est neutre. Pour s'en convaincre, il est notoire qu'en Europe, certains fichiers (manuels à l'époque) créés vers 1938 ont pu être utilisés, vers 1942, à des fins non prévues par leurs auteurs.

6.4 La loi sur la propriété des logiciels

La loi du 3 juillet 1985 portant sur les droits d'auteur comporte des dispositions sur la propriété des logiciels. Pour résumer, cette loi comporte deux points essentiels :

1. tout logiciel écrit et mis au point dans l'exercice de ses fonctions par un informaticien employé par une société, appartient à l'employeur.
2. en ce qui concerne les logiciels, « *toute reproduction autre que l'établissement d'une copie de sauvegarde par l'utilisateur ainsi que toute utilisation d'un logiciel non expressément autorisée par l'auteur ou ses ayants droit* » est passible de sanctions.

Ces dispositions ont déjà fait couler beaucoup d'encre et nous ne faisons, à leur sujet, qu'une seule remarque : la production de logiciel coûte cher et, pour trouver des logiciels de qualité à bas prix, il faut bien qu'un nombre

raisonnable de personnes les achète. En conséquence, tout informaticien souhaitant faire son métier de l'écriture de logiciel devrait être intéressé, au premier chef, à ce que le piratage reste marginal.

6.5 Loi « Godfrain », relative à la fraude informatique.

La loi Godfrain vise ce qui est communément appelé le « piratage » des systèmes informatiques (ne pas confondre avec le « piratage » des logiciels), c'est à dire l'utilisation frauduleuse d'un système de traitement automatisé de l'information. Dans ce cadre, il est interdit:

- d'accéder à un système informatique par des moyens frauduleux,
- de fausser ou d'empêcher un fonctionnement normal,
- d'y introduire, supprimer ou modifier des données.

Par ailleurs, cette loi permet de réprimer la falsification de documents informatisés.

Les peines encourues sont lourdes: emprisonnement (jusqu'à 5 ans) et amendes (jusqu'à 2 000 000 F).

6.6 Loi relative à la cryptologie

Un court extrait de la loi relative à la cryptologie permet d'en saisir les motivations:

On entend par prestations de cryptologie toutes prestations visant à transformer à l'aide de conventions secrètes des informations ou signaux clairs en informations ou signaux inintelligibles pour des tiers, ou à réaliser l'opération inverse, grâce à des moyens, matériels ou logiciels conçus à cet effet.

Pour préserver les intérêts de la défense nationale et de la sécurité intérieure ou extérieure de l'Etat, la fourniture, l'exploitation ou l'utilisation de moyens ou de prestations de cryptologie sont soumises :

a) à déclaration préalable lorsque ce moyen ou cette prestation ne peut avoir d'autre objet que d'authentifier une communication ou d'assurer l'intégralité du message transmis ;

b) à autorisation préalable du Premier ministre dans les autres cas.

6.7 Conclusions

La conclusion sera brève : « *nul n'est sensé ignorer la loi* » et nous encourageons vivement tout étudiant à consulter, dans le Journal Officiel, le texte des lois qui viennent d'être évoquées. Le Journal Officiel étant parfois d'accès difficile, nous conseillons au lecteur de faire quelques recherches sur

l'Internet: de nombreux serveurs *Web* donnent des extraits de ses textes, accompagnés le plus souvent de commentaires pratiques et de liens sur les législations étrangères.

Table des matières