Informatique

Synthèse de cours & exercices corrigés

Création de bases de données

- Toutes les étapes de la création d'une base de données, de l'analyse à la réalisation à l'aide du langage SQL
- Les mécanismes de préservation et de sécurisation des données
- Des ressources sur www.pearson.fr : fichiers des données et textes des requêtes

collection Synthex

PEARSON Education

Nicolas Larrousse

Informatique

Synthèse de cours & exercices corrigés

Création de bases de données

Nicolas Larrousse

CNRS

Avec la contribution de Éric Innocenti
Université de Corse Pasquale Paoli

collection
Synthex

PEARSON
Education
France

ISBN : 978-2-7440-7386-1
ISSN : 1768-7616

Composition sous FrameMaker : IDT

Sommaire

L'auteur

Nicolas Larrousse est ingénieur au CNRS. Spécialisé en informatique, il enseigne les bases de données au niveau Master à l'université de Versailles Saint-Quentin-en-Yvelines et au service de formation permanente de l'université Pierre et Marie-Curie à Jussieu. Il est responsable d'un atelier sur les outils informatiques pour le master de sciences cognitives de l'École normale supérieure (Ulm). Il a mis en place une formation de type IUP (bac + 4) en informatique à l'université d'Antananarivo (Madagascar). Il a participé à la conception et à la mise en œuvre de nombreuses bases de données, essentiellement dans le domaine documentaire à l'INIST (CNRS). Il est impliqué dans le programme des formations « TRANSFER » de l'AUF (Agence universitaire de la francophonie), où il s'occupe plus particulièrement des formations réseaux et des certifications Linux.

Le relecteur

Éric Innocenti est maître de conférences en informatique à l'université de Corse Pasquale Paoli. Il est responsable pédagogique des filières SRC (*Services et Réseaux de Communication*) et LPM (*Licence Professionnelle Multimédia*). Il enseigne l'algorithmique, la programmation ainsi que les systèmes d'information à l'Institut universitaire de technologie. Son parcours professionnel l'a conduit de la gestion informatique – pour le compte de sociétés privées – à la recherche universitaire – où il travaille sur la modélisation et la simulation informatique des systèmes complexes. Il est également auteur de progiciels de gestion et continue d'exercer la fonction d'analyste consultant auprès d'entreprises et d'administrations.

Avant-propos

Objectifs de l'ouvrage

Le but de cet ouvrage est de proposer une méthode à ceux qui veulent concevoir un système d'information robuste et évolutif en évitant les écueils classiques aboutissant à des données inutilisables. En effet, une mauvaise conception de départ conduit à stocker des données inutiles (redondance) et ainsi à générer des incohérences. Par ailleurs, une structure de données inadaptée peut provoquer des erreurs fondamentales d'expression dans l'interrogation de la base de données.

Il n'est pas aisé de présenter dans un seul ouvrage toutes les facettes du monde des bases de données – de l'enquête préalable à la réalisation pratique en SQL – et il a donc fallu opérer certains choix pour ne conserver que l'essentiel. La bibliographie proposée permet d'approfondir les différents sujets présentés dans l'ouvrage. La langue utilisée est volontairement peu technique pour rendre accessibles les concepts au public débutant visé. L'ouvrage s'adresse à des étudiants, de toutes les filières, qui débutent dans ce domaine, mais aussi aux professionnels qui veulent mettre en place une base de données, même de taille modeste. En effet, les problèmes pratiques que pose la réalisation d'une base de données se retrouvent à toutes les échelles, et ces aspects sont traités dans cet ouvrage au même niveau que les notions théoriques.

Organisation de l'ouvrage

Le chapitre 1 est une introduction à la notion de base de données et aux métiers associés. Il propose une mise en perspective de l'évolution des bases de données, mais aussi de leur utilisation (fouille de données, entrepôts de données, etc.).

Les chapitres 2, 3 et 4 décrivent les différentes étapes de la conception d'une base de données : la construction du modèle conceptuel (entité-association dans cet ouvrage), le passage au modèle relationnel puis la réalisation avec le langage SQL.

Le chapitre 5 reprend les notions présentées dans les chapitres précédents en les appliquant à un exemple concret, ainsi traité de manière complète. Un regard critique sur la modélisation effectuée conduit à la remise en cause et à l'évolution du modèle.

Le chapitre 6 est consacré à la préservation des données. En effet, une fois le processus de création réalisé, on doit mettre en œuvre des outils pour garantir la pérennité des données. Après avoir énoncé quelques conseils généraux, on présente ici les deux outils fondamentaux que sont les transactions et les déclencheurs (triggers).

Notation

Les mots importants d'une section sont mis en exergue par l'utilisation de gras. Lorsque l'on fait référence à des objets définis dans l'ouvrage, par exemple une table, ils sont entre guillemets simples (‘). Afin que la lecture soit facilitée et les confusions évitées, un mot unique par chapitre a été choisi pour désigner les synonymes que constituent les termes « attribut », « champ » et « colonne ». Le terme « attribut » est utilisé dans le chapitre 2 pour le modèle entité-association, le terme « champ » est utilisé dans le chapitre 3 pour le modèle relationnel et le terme « colonne » est utilisé dans le chapitre 4 consacré à SQL. Les noms des entités, des associations et de leurs attributs sont accentués dans le chapitre 2 alors que les noms des tables, des relations et de leurs champs ne sont pas accentués dans les chapitres 3 et 4 car ils sont utilisés directement dans le code SQL ainsi que dans les SGBD, qui ne les acceptent pas systématiquement.

Les mots clés du langage SQL sont en majuscules pour les différencier facilement des parties « variables » que sont les noms des colonnes et des tables qui sont en minuscules. Dans la mesure du possible, les exemples en SQL sont indépendants du SGBD employé.

Ressources complémentaires

Les scripts présentés dans l'ouvrage sont accessibles sur le site de Pearson Education France (http://www.pearsoneducation.fr), éditeur de l'ouvrage. Vous y trouverez également des jeux de données pour les bases de données que l'on utilise ici ainsi que les scripts de création des différentes tables.

Remerciements

Je tiens à remercier Éric Innocenti pour son implication dans le projet. Je remercie particulièrement Christophe Lenne, chez Pearson Education France, notamment pour sa patience.

Chapitre 1

Introduction aux bases de données

Au cours des dernières années, les bases de données ont connu un développement considérable, au point qu'elles jouent désormais un rôle dans chacune de nos opérations quotidiennes – du simple achat effectué avec sa carte bancaire jusqu'à nos déclarations de revenus.

L'objectif de ce chapitre est de définir la notion de base de données ainsi que les principaux concepts qui s'y rattachent. La méthodologie qui permet de les concevoir, les applications informatiques associées à leur mise en œuvre (SGBD) et les différents métiers des bases de données y sont présentés.

1 Qu'est-ce qu'une base de données ?

Le nombre d'informations disponibles et les moyens de les diffuser sont en constante progression. La croissance du *World Wide Web* a encore accru ce développement, en fournissant l'accès à des bases de données très diverses avec une interface commune. Celles-ci se situent au cœur de l'activité des entreprises, des administrations, de la recherche et de bon nombre d'activités humaines désormais liées à l'informatique.

Dans le domaine purement informatique, elles interviennent dorénavant à tous les niveaux. Les développeurs d'applications s'appuient sur des bases de données externes pour gérer leurs données alors qu'auparavant elles étaient intégrées dans le programme. Citons un autre exemple : la gestion des fichiers dans les nouveaux systèmes d'exploitation (par exemple, Panther de chez Apple™ ou le futur Vista de Microsoft™) évolue vers de véritables bases de données mises à jour en permanence. Elles permettent de retrouver les fichiers instantanément, par leur nom mais aussi par leur contenu, à la manière d'un moteur de recherche.

Les **bases de données** reposent sur des théories solides et sont à l'origine d'une des plus importantes disciplines de l'informatique : l'**ingénierie des systèmes d'information**. Cette section présente une idée intuitive de ce qu'est une base de données, de son utilisation puis des éléments de qualité qui lui sont associés.

1.1 Notion de base de données

Tout le monde a une idée naturelle de ce que peut être une base de données : elle peut revêtir la forme d'une liste de CD contenant le nom des artistes et les titres des morceaux ou encore celle de fiches de recettes de cuisine.

On remarque qu'une caractéristique des données contenues dans une base de données est qu'elles doivent posséder un lien entre elles. En effet, des données choisies au hasard ne constituent certainement pas une base de données. Celle-ci est donc une représentation partielle et (très) simplifiée du monde réel, que l'on a obtenu par un processus de **modélisation**. En résumé, on définit une base de données comme l'ensemble des données stockées. Pour les manipuler, on utilise généralement un logiciel spécialisé appelé SGBD (*Système de Gestion de Bases de Données*). Il y a parfois confusion, par abus de langage, entre base de données et SGBD. On appelle aussi « système d'information » l'ensemble composé par les bases de données, le SGBD utilisé et les programmes associés. Plus formellement, on appelle ***Base de Données*** (BD) un ensemble de fichiers – informatiques ou non – structurés et organisés afin de stocker et de gérer de l'information.

1.2 Utilisation d'une base de données

La création d'une base de données recèle un but précis : elle doit permettre de retrouver de l'information par son contenu en se fondant sur des **critères de recherche**. On désire, par exemple, retrouver toutes les recettes qui nécessitent des œufs ou tous les CD qui contiennent un morceau donné.

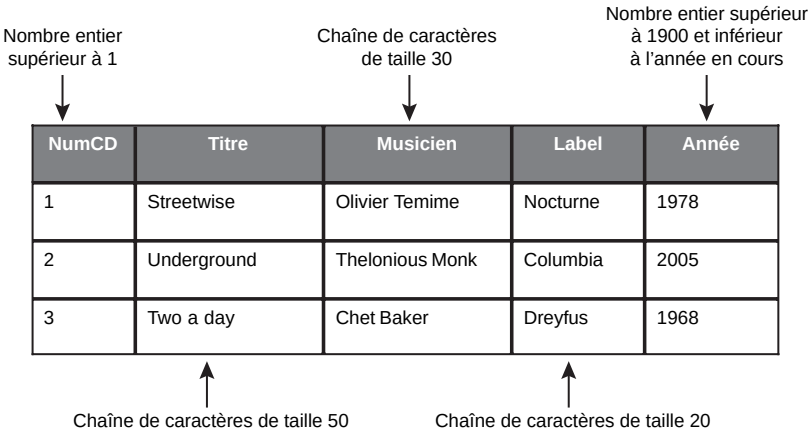
La grande différence avec un programme écrit dans un langage de programmation est qu'une base de données doit pouvoir répondre à des questions pour lesquelles elle n'a pas forcément été prévue à la conception.

Une autre différence est que les données sont susceptibles d'être utilisées par des applications différentes. Dans un programme classique, la structuration des données est décrite

directement dans le code, ce qui rend leur utilisation difficile par d'autres programmes, en particulier lorsque l'on modifie cette structure. Ce que l'on recherche en utilisant une base de données est d'assurer l'indépendance entre le traitement et les données. C'est pourquoi, il est nécessaire que l'application obtienne des informations sur la structure des données (nom, type, taille, etc.). Pour ce faire, on associe à la base de données une description que l'on appelle « **métadonnée** » ou « **catalogue** ». Cette dernière décrit la structure interne de la base de données qui est spécifique au SGBD employé (voir figure 1.1). En plus de la structure et du type des données, on stocke également à cet endroit les informations concernant les règles de cohérence des données abordées à la section suivante.

Figure 1.1

Base de données de CD et métadonnée.



L'idée générale est que l'utilisateur ou l'application utilisatrice des données ne doit pas être dépendante de leur représentation interne, ce qui constitue une abstraction des données. C'est la raison pour laquelle on utilise une description des données sous la forme d'un modèle pour permettre la restitution la plus efficace possible de l'information.

1.3 Qualité d'une base de données

Comme on l'a évoqué précédemment, l'un des objectifs de création d'une base de données est de pouvoir retrouver les données par leur contenu. Dans cette optique, il faut s'assurer que les données contenues dans la base sont de « bonne qualité ».

Comment définir la qualité des données ? De nombreux critères peuvent être pris en compte ; on peut citer parmi les principaux :

- la cohérence des données contenues dans la base ;
- l'absence de redondance.

La **cohérence des données** est fondamentale ; elle nécessite une réflexion préalable sur la normalisation du contenu des champs. On suppose qu'un champ contient la qualité d'une personne (par exemple, Monsieur, Madame, Mademoiselle). Si l'on trouve dans ce champ 'Mr' à la place de 'Monsieur', il est clair que les recherches sur ce champ par le contenu 'Monsieur' risquent d'être erronées. Dans ce cas, les informations seraient moins nombreuses que celles obtenues avec le contenu correct. On qualifie cet état de fait de « **silence** », qui signifie que certains résultats pertinents sont ignorés lors d'une interrogation.

Dans un autre cas, si l'on saisit 'Mme' pour 'Madame' et 'Melle' pour 'Mademoiselle', et qu'il y ait eu par erreur plusieurs saisies de 'Mme' alors qu'il s'agissait d'une demoiselle, la recherche par le contenu 'Mme' donne cette fois plus de résultats qu'il n'y a réellement de dames. On qualifie cet état de fait de « **bruit** », qui signifie que certains résultats non

pertinents sont retournés lors d'une interrogation. La **redondance** est parfois plus délicate à identifier. Si l'on considère le cas très simple d'un carnet d'adresses qui contiendrait en même temps le code postal et le nom de la ville, elle est ici évidente.

Tableau 1.1

Exemple de redondance de l'information.

Nom	Téléphone	Ville	Code postal
Jaco	0668541087	Bordeaux	33000
Stanley	0654789254	Nancy	54000
Marcus	0658741263	Bordo	33000
Charles	0639517720	Nancy	54000
Steve	0659874120	Boredeaux	33000

On remarque que l'on stocke plusieurs fois la même association d'information (par exemple, Nancy et 54000), ce qui consomme de la place inutilement et peut devenir significatif lorsque la base atteint quelques millions d'enregistrements.

De plus il existe des incohérences dans la saisie du nom de la ville 'Bordeaux'. La recherche par le nom 'Bordeaux' ne donnera pas le même résultat que la recherche par le code '33000'.

On verra plus loin que l'approche relationnelle procure des outils capables de détecter et d'améliorer considérablement ce genre de problèmes de qualité des bases de données.

2 Évolution des bases de données et de leur utilisation

Le développement des bases de données s'étend sur une période d'une quarantaine d'années. Cette section présente tout d'abord le contexte dans lequel elles se sont développées. On aborde ensuite les modèles utilisés successivement pour les représenter. Enfin, la dernière partie présente une vue prospective des changements dans l'utilisation des bases de données.

2.1 Contexte

À partir des années 1960, les ordinateurs évoluent rapidement. Ils sont de plus en plus performants mais aussi de plus en plus répandus du fait de leur coût plus raisonnable. Leur utilisation change également ; on passe de la notion de calculateurs purs à des machines capables aussi de traiter de l'information. On parvient à un niveau d'abstraction supplémentaire par rapport aux machines et on obtient en conséquence une indépendance par rapport à l'architecture et surtout par rapport aux constructeurs.

La décennie des années 1970 est une période « faste » pour la recherche et l'innovation en informatique dont les résultats sont encore utilisés aujourd'hui. On peut utiliser des langages de programmation de haut niveau, afin de guider, voire de façonner, la démarche du programmeur (Pascal), et l'on envisage des systèmes d'exploitation indépendants de la machine employée (Unix). C'est également à cette époque que l'on pose les fondements des techniques qui sont utilisées dans les réseaux (TCP/IP), en particulier pour Internet. C'est dans ce contexte favorable que E. F. Codd définit et développe l'approche relationnelle en base de données.

L'objectif principal est d'éloigner l'utilisateur des détails d'implémentation et de faciliter ainsi l'usage de l'informatique. Un autre but est de rendre « **génériques** » et réutilisables les développements informatiques, parfois devenus caducs en raison d'un changement de machine. Dans le domaine des bases de données, le développement de l'**architecture à trois niveaux** constitue une première étape importante. Les fonctionnalités des systèmes de bases de données sont séparées en trois niveaux : **niveau physique**, **niveau logique** et **niveau externe**.

2.2 MODÈLES

Les **modèles de données** correspondent à la manière de structurer l'information dans une base de données. Ils reposent sur les principes et les théories issus du domaine de la recherche en informatique et permettent de traduire la réalité de l'information vers une représentation utilisable en informatique.

Modèle hiérarchique et modèle réseau

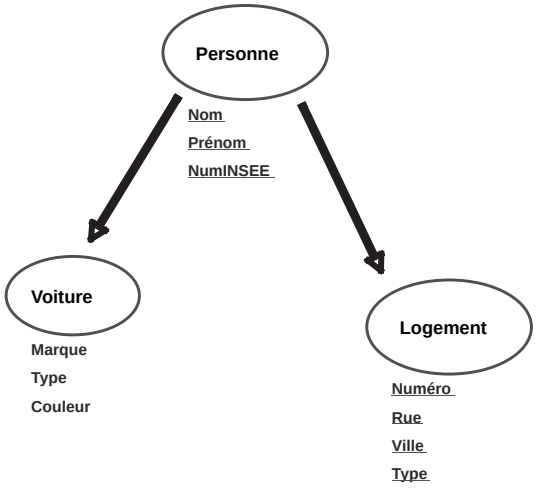
Le traitement de l'information à cette époque est encore très lié à l'organisation des fichiers sur une machine. Les modèles conceptuels de données sont eux aussi très proches du système de fichiers puisque l'on manipule des graphes ou des arbres. Les nœuds de ces structures constituent les informations et les liens entre ces données les arêtes. À ce moment, on n'est pas encore capable de séparer complètement le **niveau logique** du **niveau physique** d'un système de bases de données (voir figure 1.2).

Le modèle « hiérarchique » propose une classification arborescente des données à la manière d'une classification scientifique. Dans ce type de modèle, chaque enregistrement n'a qu'un seul possesseur ; par exemple, une commande n'a qu'un seul client. Cependant, notamment à cause de ce type de limitations, ce modèle ne peut pas traduire toutes les réalités de l'information dans les organisations.

Le modèle « réseau » est une extension du modèle précédent : il permet des liaisons transversales, utilise une structure de graphe et lève de nombreuses limitations du modèle hiérarchique. Dans les deux cas, les enregistrements sont reliés par des pointeurs : on stocke l'adresse de l'enregistrement auquel il est lié. Des SGBD de type hiérarchique ou réseau sont encore employés pour des raisons d'efficacité lorsque la structure des données s'y prête. On utilise à cet effet des SGBD de conception ancienne, comme IMS (Bull) pour le modèle réseau ou IDMS (Computer Associate).

Figure 1.2

Modèle hiérarchique.



Modèle relationnel

En 1970, E. F. Codd propose un nouveau modèle « relationnel » dans un article resté célèbre : « A Relational Model of Data for Large Shared Data Banks », *CACM 13*, n° 6, June 1970. Il cherche à créer un langage d'interrogation des bases de données plus proche du langage naturel. Dans cette optique, il fonde sa recherche sur des concepts mathématiques rigoureux, tels que la théorie des ensembles et la logique du premier ordre. Le modèle relationnel permet de modéliser les informations contenues dans les bases de données en utilisant des **relations**, c'est-à-dire des ensembles d'**attributs** (voir figure 1.3).

De l'idée de départ à la réalisation d'un produit utilisable, le laps de temps est souvent de l'ordre d'une décennie. La mise en œuvre des idées de Codd se fait chez IBM dans le cadre du projet de recherche System-R. Le premier produit commercial sera non pas le fait d'IBM, mais celui d'Honeywell en 1976. Il sera suivi d'un produit réellement abouti de chez Relationnel Software en 1980 : Oracle, qui a connu le succès que l'on sait. De son côté IBM en tirera un produit qui deviendra DB2.

Toujours dans le cadre du projet de recherche System-R, E. F. Codd met au point, en même temps que le modèle relationnel, un langage d'interrogation des données, SEQUEL, qui deviendra ensuite **SQL** (***Structured Query Language***). La normalisation du langage SQL dès 1986 par l'ANSI (institut de normalisation américaine), puis par l'ISO (organisation internationale de normalisation), a assuré pour une grande partie le succès du modèle relationnel auprès des entreprises. Fait rare dans le monde informatique, ce langage a été adopté par la quasi-totalité des éditeurs commerciaux qui participent activement à son évolution. SQL est devenu le standard de fait, même si aucun éditeur ne respecte à la lettre la norme. D'ailleurs, à partir de SQL 2, il existe une définition de quatre niveaux de compatibilité avec la norme officielle. La normalisation de ce langage garantit sa pérennité, même si son évolution s'en trouve ralentie. Les requêtes écrites pour un SGBD fonctionnent en général sans trop de modifications avec un autre SGBD, ce qui permet d'envisager des migrations moins douloureuses et de conserver une partie de l'investissement initial.

Figure 1.3

Modèle relationnel.

Ouvrage (Cote, Titre, Auteur,Editeur)

Cote	Titre	Auteur	Editeur
1	La programmation sous Unix	J. M. Rifflet	MC Graw Hill
2	Les bases de données	G. Gardarin	Eyrolles
3	Internet pour les nuls	C. Baroudi	First Interactive
4	Le langage C	C. Delannoy	Eyrolles
5	Petit Larousse illustré	P. Larousse	Larousse
6	Godel, Escher & Bach	D. Hofstadter	Basic Book Inc.

Modèle objet

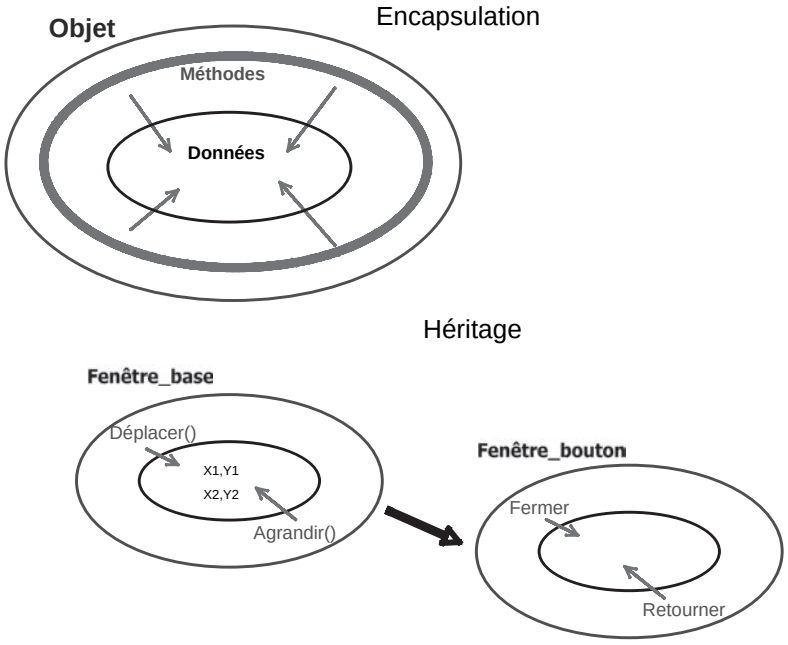
Dans le sillage du développement des langages orientés objet (C++, Java…) dans les années 1980, le **concept objet** a été adapté aux bases de données. Plusieurs raisons, en dehors des qualités reconnues de l'approche objet, ont conduit à définir une extension objet pour les bases de données (voir figure 1.4).

La première est que le modèle relationnel, dans sa simplicité, ne permet pas de modéliser facilement toutes les réalités. La deuxième est qu'un objet permet de représenter directement un élément du monde réel. Les structures d'éléments complexes se retrouvent souvent dispersées entre plusieurs tables dans l'approche relationnelle classique. De plus, le concept objet est mieux adapté pour modéliser des volumes de texte importants ou d'autres types de données multimédias (sons, images, vidéos…). Enfin, il est beaucoup

plus commode de manipuler directement des objets lorsque l'on développe avec un langage à objet (comme C++ ou Java). Les bases de données, on le rappelle, sont dorénavant des briques constitutives des applications. Les **bases de données « orientées objet »** apportent ainsi aux applications développées en langage objet la **persistance** des objets manipulés : ces derniers peuvent ainsi directement être réutilisés par l'application d'origine ou par d'autres sans redéfinition. Ces concepts ont été intégrés à partir de la version 2 de la norme SQL.

Les produits commerciaux adaptés à ces concepts n'ont pas connu une diffusion suffisamment importante. Le monde des bases de données évolue assez lentement : la migration d'un système d'information vers l'objet représente pour une organisation un investissement considérable qui n'est pas toujours justifié. La robustesse et la popularité de l'approche relationnelle, qui a mis presque vingt ans à s'imposer, a également freiné le développement de l'approche objet pure dans les bases de données. Les données modélisées sous forme d'objets sont aussi plus complexes à représenter du point de vue du SGBD et l'on rencontre encore très souvent des problèmes de performance.

Figure 1.4
Modèle objet.



Modèle relationnel-objet

Une demande d'évolution du strict modèle relationnel existe toutefois. En effet, la gestion des données autres que du texte et des nombres – comme des images, du son et des vidéos – implique l'évolution du modèle relationnel. De même, les champs dits « multivalués », disposant de plusieurs valeurs telles qu'une liste de prénoms ou des coordonnées géographiques, ne peuvent pas être modélisés efficacement en utilisant ce type d'approche. L'idée est alors d'intégrer de l'objet au modèle relationnel existant plutôt que d'utiliser l'approche objet pure. Il convient de remarquer que ce type d'évolution a déjà été développé dans le cadre des langages de programmation. Le langage C++ est l'évolution intégrant l'approche objet du langage C et non pas un langage à objet pur comme peut l'être Smalltalk.

Cette extension, adoptée par la plupart des SGBD, se nomme « **relationnel-objet** » et permet aux concepteurs des bases de données de disposer de types évolués « abstraits » plus simples à concevoir et surtout plus commodes à faire évoluer. Elle offre en outre la possi-

bilité de modéliser plus facilement la complexité des organisations (voir figure 1.5). Dans cette optique, la norme SQL a logiquement été adaptée. Dans sa version 3, elle prend en compte l'extension objet. Les types de données sont étendus et les opérations d'encapsulation et d'héritage, typiques de l'approche objet, sont supportées. Cette solution a l'avantage d'offrir un bon niveau de compatibilité avec l'approche précédente très répandue et d'effectuer ainsi une migration plus aisée.

Figure 1.5

Modèle relationnel-objet.

Cote	Titre	Auteur	Editeur
1	Le vide	Nom: Durand, Atamp / Prénom: Dal, Charles	Les éditions du temps qui ne passe pas
2	La vie sans mode d'emploi	Nom: Brassé, Duporche, Château, Paclaire / Prénom: Alexis, Jean-Marie, Romain, Anne-Isabelle	Romazava

2.3 ÉVOLUTION DE L'UTILISATION DES BASES DE DONNÉES

Cette section présente une description de nouvelles manières de stocker ou d'utiliser les bases de données. La différence par rapport à la section précédente est que l'on ne remet pas en cause le modèle utilisé pour décrire les données sauf, dans une certaine mesure, pour le cas de XML.

Base de données réparties

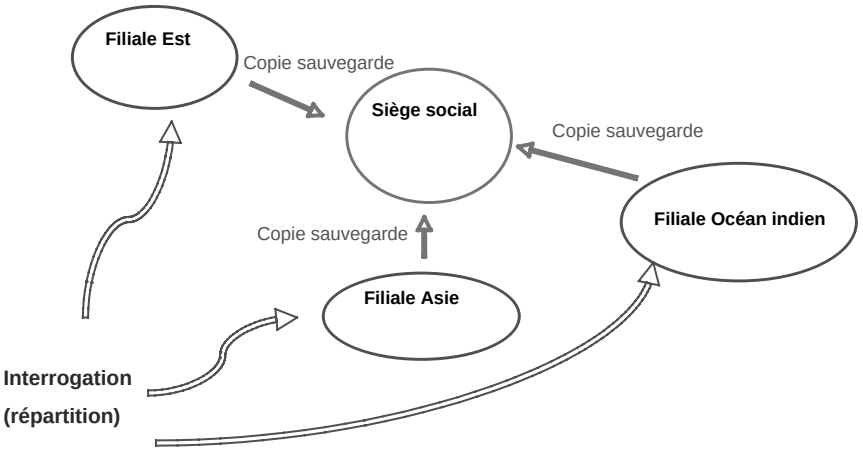
Le déploiement des réseaux ainsi que l'augmentation de leur débit ces dernières années ont conduit à répartir les données sur plusieurs sites géographiques, ce qui facilite la politique de décentralisation des organisations. Ce type d'architecture masque la répartition de l'information tout en garantissant une gestion transparente aux utilisateurs, comme s'ils disposaient d'une seule base de données (voir figure 1.6). Les bases de données réparties assurent ainsi une plus **grande fiabilité**, de **meilleures performances** et facilitent l'**évolution du système d'information**.

- **La fiabilité et la sécurité.** On effectue une copie de sécurité des données sur un site distant à intervalles réguliers pour éviter le désastre de la perte de données due à un incendie par exemple.
- **La disponibilité.** On procède à des réplications quasi permanentes des données dans le but de « rapprocher » les utilisateurs des données d'un point de vue de la topologie du réseau. On améliore également le temps de réponse en répartissant la charge entre les serveurs. Cette distribution est gérée de manière intelligente par les systèmes informatiques, ce qui permet de répartir l'information efficacement sur les différents sites, en fonction des accès utilisateurs. Ces principes sont notamment utilisés par les moteurs de recherche.
- **Les données sont réparties sur des sites séparés.** Le dispositif permet de masquer cet aspect aux utilisateurs et de fonctionner comme si un seul serveur était présent sur un seul site. L'évolution du système est rendue totalement **transparente** pour les utilisateurs : en cas notamment de changement de machine à la suite d'une panne, de

modification de localisation, d'augmentation de la taille de la base, d'ajouts d'ordinateurs sur le réseau afin d'augmenter la capacité de stockage de la base de données.

Figure 1.6

Base de données réparties.



Ces technologies possèdent néanmoins des inconvénients. La sécurité sur les réseaux informatiques nécessite beaucoup plus de travail que dans le cas d'un système non réparti. Les techniques de sécurité à mettre en œuvre sont plus complexes et plus coûteuses.

Extraction d'informations non explicitement stockées dans une base de données

Il existe deux manières de « créer » de la nouvelle information à partir de l'information stockée explicitement dans une base de données. Soit on est capable d'énoncer des règles précises de constitution de l'information à partir du sens même des données, soit on utilise des méthodes d'analyse afin de trouver des corrélations sur un volume de données important, ce qui permet ensuite d'en déduire des règles.

La première possibilité a été formalisée et mise en œuvre sous le nom de **base de données déductives**, dans le but d'utiliser des méthodes semblables à celles pratiquées pour la déduction en intelligence artificielle. On définit un ensemble de règles et on les applique aux données de la base, à l'aide d'un programme que l'on appelle un « moteur d'inférence ». Les SGBD qualifiés de déductifs utilisent à cet effet un dérivé du langage Prolog : **Datalog**.

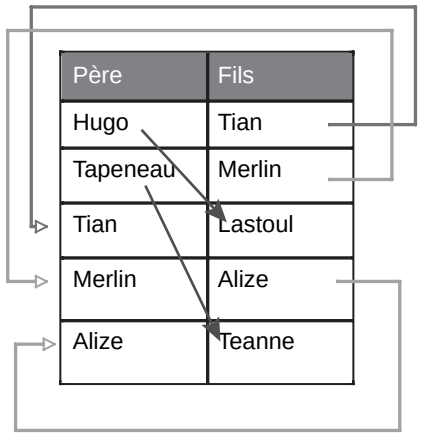
Les relations de parenté entre individus sont une bonne illustration de l'utilisation des bases de données déductives. Intuitivement, on peut appréhender le fonctionnement général de ces SGBD déductifs en considérant l'exemple suivant. On suppose que l'on a une base de données qui représente les relations « père-fils » (voir figure 1.7). Une règle très simple permet de définir un lien de parenté « ancêtre » :

Si père-fils(X,Y) et père-fils(Y,Z), alors ancêtre(X,Z).

La seconde possibilité d'obtenir l'information nouvelle à partir de l'information existante relève des méthodes dites de « **fouilles de données** » (ou **data mining**). Ce type d'exploitation des bases de données a connu un grand succès ces dernières années, par l'analyse de grands volumes de données afin d'identifier des corrélations entre des valeurs de champs. Par exemple, « les personnes moustachues de plus de quarante ans habitant une maison individuelle lisent plutôt des revues de psychologie ». Les techniques d'analyses employées sont une alchimie concoctée à partir de statistiques, de réseaux de neurones, de classifications et autres techniques employées en intelligence artificielle. Une fois ces « règles » établies, on peut les utiliser au sein des bases de données déductives décrites précédemment.

Figure 1.7

Base de données déductives père-fils.



L'information décisionnelle ainsi extraite a de nombreux débouchés : du ciblage marketing à la médecine en passant par la prévision financière. Les méthodes s'affinent et deviennent réellement efficaces, ce qui a cependant donné lieu à quelques escroqueries, les entreprises spécialisées dans le domaine refusant bien sûr de donner les spécifications de leurs méthodes d'analyses, qui relevaient parfois de la divination, au motif de ne pas perdre leur avantage concurrentiel.

Entrepôts de données (datawarehouse)

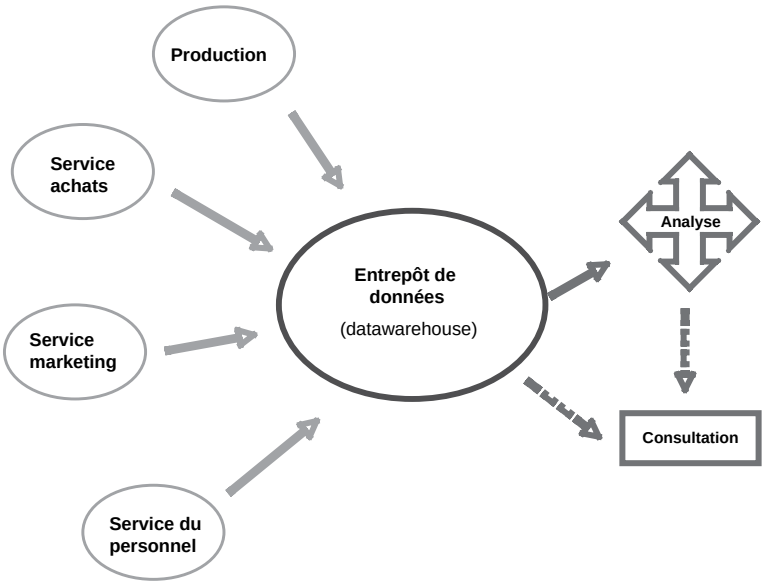
Les entrepôts de données sont des bases de données « récapitulatives », constituées à partir de différentes sources de l'entreprise (comptabilité, ventes, achats, service du personnel…) pour disposer d'un accès homogène à l'ensemble des données. Les données dispersées ne peuvent pas être exploitées directement en tant qu'information décisionnelle. En effet, il n'est pas possible d'analyser efficacement dans le temps des indicateurs de gestion par métiers ou par individus. Les données sont alors rapatriées et stockées dans un **entrepôt de données** ou **datawarehouse** (voir figure 1.8).

Si l'information est disponible, on stocke les différentes valeurs d'une donnée, ce qui permet de conserver un historique dans le temps de certaines données : elles sont dites **historisées**. Ce dernier aspect est désigné parfois aussi comme une **base de données temporelle**, dans la mesure où le SGBD procure des outils d'analyse adaptés. Il est alors possible d'effectuer des analyses décisionnelles qui combinent tous les paramètres de l'entreprise : c'est l'**analyse décisionnelle multidimensionnelle**. Les techniques précédentes de fouille de données et de déduction peuvent être utilisées. Elles sont combinées avec d'autres outils d'analyse de l'évolution d'une donnée à partir de son historique pour en extrapoler des tendances et ainsi faire des prévisions.

L'idée des entrepôts de données est plus qu'intéressante, mais il est évident que sa mise en œuvre est délicate tant d'un point de vue du stockage de l'information (volume de données, hétérogénéité des données à traiter…) que de l'analyse des données (comment analyser, quelle information stocker…). En outre, il s'agit le plus souvent du poste budgétaire le plus onéreux dans un projet d'informatique décisionnelle.

Figure 1.8

Entrepôts de données.



XML

Le *World Wide Web* est la mise en œuvre du concept d'hypertexte par l'utilisation de la structure de communication fournie par Internet. Les fichiers dispersés sur différentes machines du réseau Internet peuvent ainsi être associés. Son succès a provoqué un glissement de l'idée originale, qui consistait à relier simplement des textes entre eux, vers la notion d'interface universelle. On ne transmet plus seulement le contenu d'un fichier statique, mais également le résultat de l'exécution d'un programme : le contenu devient donc dynamique. Par extension, on imagine aisément que le mécanisme du Web peut également transmettre le résultat de l'interrogation d'une base de données.

Le concepteur du Web, T. B. Lee, s'est appuyé pour sa mise en œuvre, outre la structure technique existante d'Internet, sur un langage de description de documents utilisant des balises : le SGML (*Standard Generalized Markup Language*). Le découpage du document par les balises est décrit dans un document associé que chacun peut créer en fonction des ses besoins : la DTD (*Data Type Definition*). Cette dernière, formulée dans un langage normalisé, permettra à des programmes spécialisés, les « parsers », de vérifier si le document est conforme à la structure attendue et d'en extraire facilement le contenu. Le langage SGML est assez complexe à manipuler, car il est très général. Pour des besoins spécifiques, on utilise des versions simplifiées, telles que HTML ou XML.

T. B. Lee a donc développé un langage de présentation, HTML (*Hyper Text Markup Language*), basé sur les notions développées par SGML. Ce langage permet essentiellement de spécifier des critères de présentation (gras, souligné, couleur…) et, bien sûr, de décrire les liens entre les fichiers. Il ne comprend aucun élément de structuration des données du texte contenu dans la page HTML. Les moteurs de recherche parcourent le Web et indexent le contenu des pages par rapport à des listes de mots clés combinés à d'autres méthodes (beaucoup) plus sophistiquées. Cependant, ils ne peuvent différencier dans le texte le titre d'un résumé ou d'une légende associée à une image. L'efficacité et la pertinence de l'indexation en sont diminuées d'autant.

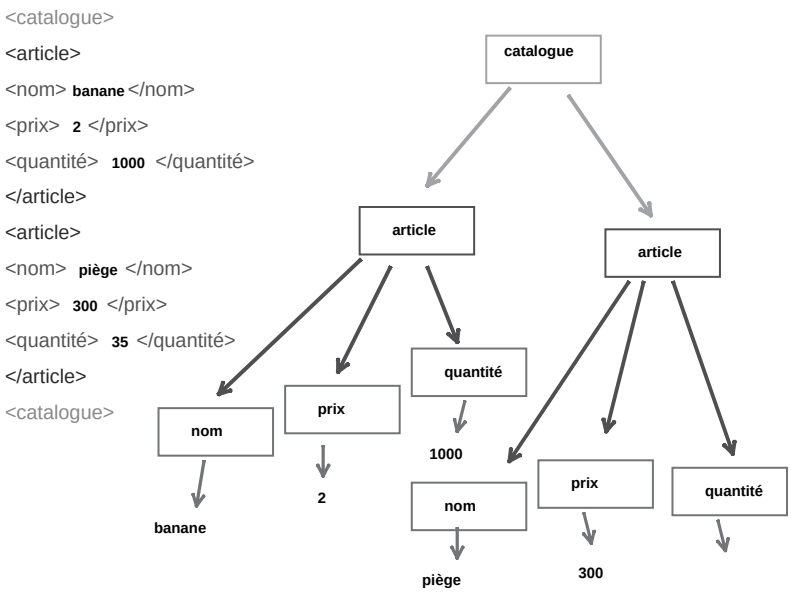
Pour remédier à cela, le W3C (*World Wide Web Consortium*) a défini un langage, qui est également une simplification de SGML, permettant de décrire la structure interne d'un document : XML (*eXtended Markup Language*). La structure d'un document XML est représentée sous la forme d'un arbre tout comme celle d'un document HTML. La description de

cette structure arborescente se trouve dans une DTD ou plus récemment dans un schéma XML. Le schéma est plus souple et permet d'employer les mêmes outils de traitement que pour les fichiers XML. Le langage HTML possède évidemment lui aussi une DTD, mais, à la différence de XML, elle est normalisée par le W3C et ne peut être modifiée et adaptée pour ses besoins propres (voir figure 1.9).

L'objectif à terme est que les fichiers du *World Wide Web* soient désormais décrits en utilisant XML à la place de HTML. La présentation des données repose alors sur les « feuilles de styles » (telles que *eXtended Stylesheet Language*) pour générer les formats de présentation classiques (HTML, PDF, PostScript…) à partir du format XML. De cette manière, les moteurs de recherche pourront extraire directement par exemple le résumé ou le titre d'un paragraphe d'un fichier XML. L'indexation peut ainsi être plus précise ; un mot figurant dans un titre étant plus important que le même mot dans une note de bas de page.

Figure 1.9

Structure arborescente XML.



Quel est le rapport avec les bases de données ? Comme on l'a vu précédemment, une page Web peut être le résultat d'une requête provenant d'un SGBD ; c'est même devenu le moyen le plus courant d'interroger un SGBD. Dans cette optique, si le SGBD est capable de générer directement du XML, cela facilite le processus. C'est d'autant plus vrai que le passage du modèle relationnel à un modèle arborescent de type XML est parfois complexe et qu'il est bien agréable que le SGBD sache le faire.

Le langage de description XML, par sa versatilité, s'impose comme un format d'échange universel. C'est évident pour des fichiers générés par un traitement de texte : on sépare ainsi l'aspect structurel de l'aspect présentation. On se donne également la possibilité d'ouvrir le document indépendamment du logiciel utilisé, ce qui garantit sa pérennité. De la même manière, XML est utilisé comme format d'échange entre SGBD et d'un SGBD vers d'autres logiciels (tableurs, traitements de texte…).

Le langage XML est adopté petit à petit par la plupart des éditeurs et il est amené à jouer un rôle croissant dans les échanges de données. De plus en plus de SGBD sont capables de produire des résultats de requête en XML, d'importer du XML et acceptent même de gérer les données directement dans ce format. Cette dernière possibilité implique que les SGBD supportent des données moins bien structurées : cette capacité constitue l'une des évolutions futures des SGBD.

Contenu multimédia

Le multimédia s'est fortement développé depuis la fin des années 1990. La demande étant très forte dans ce domaine, les éditeurs de SGBD se doivent d'intégrer de l'information multimédia (image, son, vidéo…) dans les bases de données. Les **bases de données multimédias** posent de nouveaux problèmes, en particulier pour effectuer des recherches sur les contenus multimédias, ce qui est par nature difficile.

Une solution est d'effectuer une indexation préliminaire manuelle à l'aide de mots clés qui permettent d'opérer par la suite des interrogations, mais cela semble illusoire de réaliser ce traitement pour des volumes importants de documents multimédias. Dans le cas contraire, il existe des méthodes de recherche sur des fichiers de type image par rapport à des schémas prédéfinis. Cette possibilité reste pour l'instant plutôt du domaine de la recherche, même si l'on est déjà (malheureusement) capable d'identifier des visages par rapport à un modèle dans certaines conditions.Dans le même ordre d'idée, il est déjà possible d'utiliser des techniques pour évaluer le style de musique (classique, jazz, pop…) d'un fichier en analysant son contenu. On est cependant assez loin de pouvoir identifier un genre de film en se basant sur l'analyse des images.

Les bases de données multimédias constituent un sujet de recherche très prometteur et très actif. Ces problématiques relèvent pour l'instant plutôt des préoccupations des sociétés développant les moteurs de recherche que des bases de données au sens strict. Cependant, le groupe de normalisation ISO/IEC prépare l'évolution pour le multimédia de la norme SQL-MM, qui est une évolution pour le multimédia de la norme SQL.

3 Systèmes de gestion de bases de données

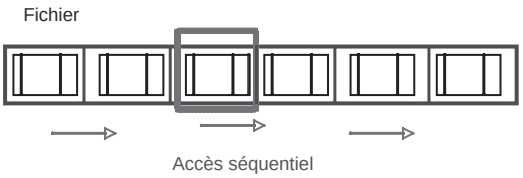
On a beaucoup parlé dans les sections précédentes des SGBD. Un SGBD est un logiciel complexe qui permet de gérer et d'utiliser les données que l'on stocke en utilisant les modèles cités précédemment. La première partie de la section permet de comprendre le mécanisme d'abstraction à partir duquel se fait le passage du simple fichier informatique à la gestion de l'information qui se fonde sur l'utilisation d'un SGBD. Une seconde partie détaille le modèle en couches des SGBD ainsi que leurs fonctionnalités de base.

3.1 Fichiers informatiques

Un fichier peut être vu (historiquement) comme un morceau de bande magnétique. Les mécanismes de gestion de fichiers des langages de programmation, comme le langage C, fonctionnent encore avec cette métaphore. Lorsqu'on utilise un fichier pour stocker de l'information, il est nécessaire de prévoir un découpage de celui-ci par enregistrements, souvent de taille fixe. Pour passer d'un enregistrement à l'autre, il suffit alors d'avancer la « tête » de lecture de la taille d'un enregistrement (voir figure 1.10). Les données sont stockées dans l'enregistrement par un découpage interne suivant la taille de chaque donnée. On utilise généralement des structures de données (par exemple, en langage C) pour récupérer directement chaque valeur de champ dans une variable. Dans une base de données, on recherche les données par leur contenu. Pour retrouver l'une d'entre elles, il faut donc parcourir tous les enregistrements un à un (**recherche séquentielle**), et en examiner le contenu.

Figure 1.10

Fichier informatique.



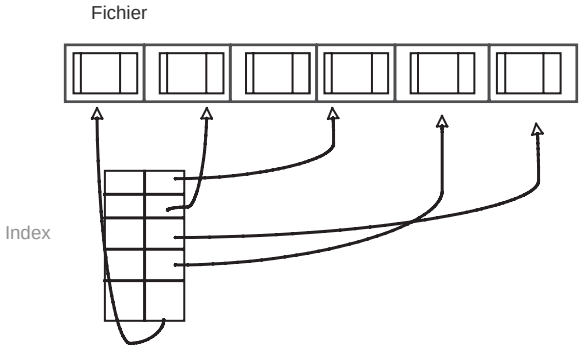
Dans le cas d'un accès séquentiel, la recherche d'un enregistrement en position n nécessite d'accéder aux n–1 enregistrements qui le précèdent. Si l'on recherche l'article de référence DR-NetCard10.102, contenu dans le fichier Article, il sera nécessaire de parcourir tous les enregistrements, depuis le début du fichier jusqu'à l'article recherché. Pour retrouver une information, il faut donc parcourir tous les enregistrements un à un et en examiner le contenu.

Une alternative au parcours séquentiel est de construire des tables descriptives afin d'accélérer l'accès aux données. Une première table permet l'**accès direct** à un enregistrement par une « clé » associée à l'adresse (pointeur) de l'enregistrement. On rappelle que c'est ce mécanisme de « pointeurs » sur des enregistrements qui est modélisé dans les modèles hiérarchiques ou réseaux pour faire le lien entre des enregistrements. On constitue ainsi le graphe qui permet de « naviguer » dans l'ensemble des enregistrements.

Une seconde table contient l'ordre relatif des enregistrements ordonnés suivant les valeurs d'un champ : on appelle cette table un **index** (voir figure 1.11). Cette seconde table permet d'employer des méthodes de recherche par le contenu du champ indexé beaucoup plus efficaces qu'une recherche séquentielle. La recherche dichotomique bien connue est l'une d'entre elles. Une fois l'enregistrement identifié, on y accède directement grâce à la première table. Les techniques de constitution des index constituent un sujet à part entière ainsi que les algorithmes de recherche qui leur sont associés.

Figure 1.11

Fichier et index.



On constate que la simple recherche d'informations recourt déjà à des algorithmes assez sophistiqués et nécessite la construction et le maintien de structures annexes pour être efficace. De même, la destruction d'enregistrements et l'évolution de la structure de la base sont également des opérations lourdes à mettre en œuvre ; elles requièrent souvent la recopie complète des informations dans un nouveau fichier. Cette section nous permet de comprendre pourquoi on utilise préférentiellement des SGBD pour gérer les données. Toutes ces fonctionnalités et bien d'autres que nous allons détailler sont intégrées dans le logiciel.

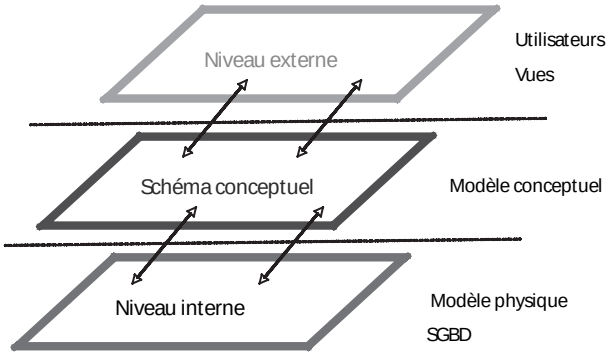
3.2 FONCTIONNALITÉS D'UN SGBD

De même que l'ISO a déterminé un modèle théorique en sept couches pour distinguer les applications réseaux et leurs interactions, il existe désormais un modèle théorique en trois couches (trois niveaux d'abstraction) afin de concevoir et d'organiser les fonctionnalités des SGBD. Ce modèle est élaboré par la commission SPARC de l'ANSI : c'est l'**architecture ANSI/SPARC** (voir figure 1.12). Cette dernière, qui date de 1975, s'inscrit dans les concepts et théories de la première génération des bases de données, dont l'objectif est d'avoir une **indépendance entre les données et les traitements** :

- Niveau **interne** ou **physique**. C'est le niveau le plus « bas ». On décrit les structures de stockage de l'information, ce qui le rend très dépendant du SGBD employé. Il se fonde sur un **modèle de données physique**.
- Niveau **conceptuel**. Il correspond à l'implémentation du **schéma conceptuel de la base de données**, que l'on réalise lors de la phase d'analyse (voir *Modèle Conceptuel des Données*, *Modèle Logique des Données*). Il est utilisé pour décrire les éléments constitutifs de la base de données et les contraintes qui leur sont associées. Il s'agit d'une certaine façon de la « documentation » de la base de données.
- Niveau **externe**. Le niveau externe sert à décrire les **vues** des utilisateurs, c'est-à-dire le **schéma de visualisation des données** qui est différent pour chaque catégorie d'utilisateurs. Un schéma externe permet de masquer la complexité de la base de données complète en fonction des droits ou des besoins des utilisateurs. Cela facilite la lecture et la sécurité de l'information.

Figure 1.12

Niveaux ANSI/ SPARC.



Il s'agit comme pour le modèle réseau de l'ISO d'un cadre de réflexion ; les SGBD ne respectent pas à la lettre le découpage proposé. Ils se doivent cependant de posséder les principales caractéristiques qui découlent de ce modèle en couches :

- **Indépendance physique des données.** Masquer la représentation interne des données ainsi que les méthodes système d'accès aux utilisateurs.
- **Indépendance logique des données.** Permettre la modification du schéma conceptuel des données sans remettre en cause les mécanismes de stockage et de manipulation internes des données.
- **Intégrité des données.** Faire en sorte que l'information résultant des liens entre les données soit cohérente.

Il peut apporter également des fonctionnalités supplémentaires utilisées dans le cadre de bases de données réparties décrites précédemment :

- **Réplication des données.** Copie automatisée de sauvegarde.

- **Virtualisation des données.** Masquage de la distribution géographique des données.
- **Haute disponibilité des données.** Duplication de la base de données sur différents sites pour diminuer la distance client/serveur et la charge des serveurs.

Le but principal de l'utilisation d'un SGBD est de masquer la représentation physique des données et les méthodes d'accès que l'on vient de voir précédemment. Cependant les mécanismes de création, d'indexation et de recherche sous-jacents sont globalement les mêmes. Évidemment, pour des questions d'efficacité, les SGBD utilisent leur propre gestion de fichiers et parfois même contournent le système de fichiers fourni avec le système d'exploitation de la machine.

Un SGBD doit permettre également la manipulation de la structure de la base de données, comme l'ajout et la modification de champs, de manière transparente. Il conserve à cet effet une description de la structure de la base de données que l'on appelle le « dictionnaire de données ». Pour réaliser ces opérations avec l'indépendance souhaitée par rapport à la représentation, le SGBD offre deux langages de haut niveau :

- un *Langage de Description de Données* (LDD) qui permet d'agir sur la structure de la base de données (ajout, suppression et modification des tables) ;
- un *Langage de Manipulation de Données* (LMD) qui permet d'interroger et de mettre à jour le contenu de la base de données.

Ces langages sont de type « non procédural », c'est-à-dire que l'on s'intéresse à l'effet de l'opération (le quoi) et non pas à la manière dont elle est réalisée (le comment). On a pu se rendre compte dans la section précédente du niveau de complexité de certaines opérations qui, grâce à ces langages, sont énoncées simplement. Par exemple, la modification de la taille d'un champ peut être énoncée en une seule instruction avec le LDD. Il est courant que les SGBD modernes implémentent ces langages de manipulation à l'aide d'objets graphiques.

Le SGBD doit également assurer la protection des données en cas de problèmes. Ceux-ci peuvent être la conséquence d'une manipulation malheureuse, mais également d'une panne du système qui survient par exemple à la suite d'une coupure de courant. Dans tous les cas, le SGBD doit permettre de restaurer les données. Ces opérations sont généralement réalisées en utilisant des « journaux » qui enregistrent au fur et à mesure les opérations faites sur la base : c'est le **mécanisme de la journalisation**. Ce journal est utilisé pour refaire, ou défaire le cas échéant, ces opérations.

En ce qui concerne les opérations de modification effectuées sur la base de données, que l'on appelle des **transactions**, des propriétés de mesure de la qualité de ces transactions sont proposées sous le terme ACID :

- **Atomicité.** Une transaction est « atomique » ; elle est exécutée entièrement ou abandonnée.
- **Cohérence.** La transaction doit se faire d'un état cohérent de la base vers un autre état cohérent.
- **Isolement.** Des transactions simultanées ne doivent pas interférer entre elles.
- **Durabilité.** La transaction a des effets permanents même en cas de panne.

À noter que tous les SGBD ne réalisent pas cette propriété ACID pour les transactions.

Les machines sont connectées au réseau ou sont simplement multi-utilisateurs : le SGBD doit permettre de donner l'accès aux bases de données à plusieurs utilisateurs concurremment. **L'accès concurrentiel** implique des opérations algorithmiques complexes à réaliser, puisqu'il faut par exemple empêcher la modification d'une valeur par un utilisateur alors qu'elle est en lecture par un autre. Cela nécessite la gestion d'une structure de description

des utilisateurs comprenant les droits qui leur sont associés pour chaque élément (lecture, modification…) : les **droits d'accès aux données**. Les mécanismes sont les mêmes que ceux qui sont mis en œuvre dans les systèmes d'exploitation multi-utilisateurs.

4 Étapes de la conception des bases de données

On peut décomposer le processus de conception d'une base de données en plusieurs étapes :

- l'analyse du système du monde réel à modéliser ;
- la mise en forme du modèle pour l'intégrer dans un SGBD ;
- la création effective dans le SGBD des structures et leur remplissage (voir figure 1.13).

4.1 Analyse du monde réel

La première étape de la démarche de modélisation des données consiste à effectuer l'analyse de la situation du monde réel à considérer. Cette action s'apparente au travail effectué par une entreprise de consulting. C'est une approche « humaine » qui se fonde en partie sur des entretiens avec les personnels concernés et ressemble plutôt à une analyse du discours et de l'organisation de l'entreprise. C'est lors de cette phase d'analyse que l'on détermine les objectifs du système d'information à concevoir et que l'on identifie tous les éléments à prendre en compte dans le système ; ce sont les champs qui contiendront les données. Un ensemble de champs peut constituer un objet du monde réel. Par exemple les champs « nom », « prénom » et « adresse » que l'on regroupe constituent une « personne ».

Enfin, il faut identifier les liens à modéliser entre ces objets ainsi que les éléments caractéristiques de ces liens. Par exemple une personne achète une voiture à 10 000 euros. Ici les deux objets liés sont « personne » et « voiture », et le prix est le composant du lien. Cette étape est délicate et fondamentale, car elle conditionne l'aspect représentatif et la qualité du modèle du monde réel considéré. Lors de cette phase, il convient également d'exprimer les règles qui définissent le domaine de validité du contenu des champs. Par exemple, le prix d'une voiture ne peut pas être inférieur à 500 euros ou supérieur à 150 000 euros.

Cette modélisation du réel permet de proposer un schéma conceptuel qui servira à la description générale du système d'information. La notion de sens des données et surtout des liens entre les entités ne sera réellement exprimée que dans ce schéma qui est plus proche du monde réel. Ce schéma est souvent réalisé à l'aide de la symbolique du modèle « entité-association » ou, plus couramment aujourd'hui, exprimé avec le langage UML (*Unified Modeling Language*). Il existe différentes méthodes intégrant les concepts présentés ci-dessus. L'objectif est de guider le travail d'analyse et d'aider à la réalisation d'un modèle de données le plus juste possible. Parmi celles-ci, la méthode Merise a connu un certain succès dans le domaine en France.

4.2 Passage au SGBD

La représentation précédente doit être transformée pour la rendre acceptable par le SGBD, qu'il soit relationnel, objet ou relationnel-objet. Souvent, cette étape modifie considérablement les objets du monde réel ainsi que les liens définis dans le schéma précédent. C'est lors de cette phase que l'on vérifie la qualité de la base de données en utilisant les critères

vus précédemment, comme l'élimination de la redondance. Le modèle relationnel procure à cette fin des outils capables de tester la cohérence du système et de le modifier le cas échéant : ce sont les « formes normales », qui seront vues au chapitre 3.

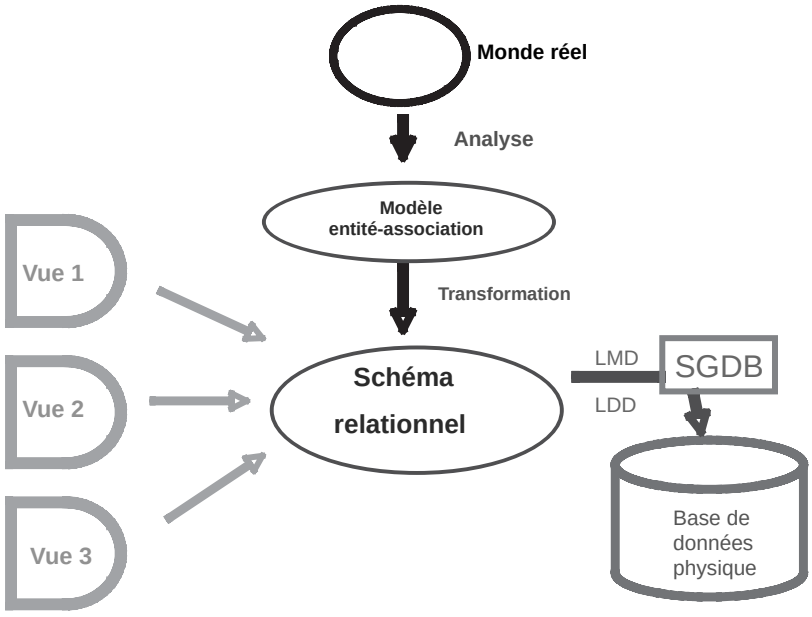
Il est possible de constater des incohérences à ce niveau de l'analyse, ce qui implique de modifier le modèle conceptuel de données développé à l'étape précédente. On obtient un schéma des données qui fournira aux utilisateurs les informations nécessaires pour effectuer leurs requêtes, par exemple la description des noms de tables, de champs et leurs types. Par contre, on perd à ce niveau l'information du « sens » des données et du lien entre elles. Ce schéma n'est guère utilisable en pratique sans le précédent. En effet, comment savoir que les personnes achètent des voitures et non pas le vendeur si l'on ne dispose pas de l'information de liaison entre les objets du monde réel ? C'est également lors de cette phase que l'on définit les « vues » du système d'information qui sont adaptées à chaque catégorie d'utilisateurs.

4.3 Création et utilisation de la base de données

Une fois le schéma précédent défini, on utilise le SGBD pour passer à la création des tables qui constituent la base de données. Puis, on insère évidemment les valeurs dans les tables. Le cas échéant, on crée les vues définies à l'étape précédente et par là même les utilisateurs concernés. Le système est alors opérationnel. Toute cette étape se fait forcément en utilisant le SGBD, alors que les précédentes étaient plus théoriques. La création des tables et l'utilisation de la base de données nécessiteront le langage SQL. Cependant, il existe de nos jours de nombreux outils graphiques dans les SGBD qui masquent l'utilisation du SQL.

Figure 1.13

Étapes de la conception d'une base de données.



5 « Métiers » des bases de données

Comme on peut le constater lorsque l'on considère les différentes étapes de la conception d'une base de données, des acteurs aux compétences très diverses interviennent dans ce processus.

5.1 Consultants/analystes

Ils prennent en charge la première étape qui consiste en l'analyse des activités et des flux d'information mis en jeu dans le monde réel à modéliser. Le profil de ces acteurs n'est pas toujours purement technique, puisque cette phase nécessite parfois beaucoup de dialogues et de psychologie pour parvenir à faire exprimer leurs besoins réels par les futurs utilisateurs. La gageure est de parvenir à faire exprimer correctement les besoins d'informatisation par les utilisateurs du système d'information, afin de proposer un modèle conceptuel de données le plus juste possible.

5.2 Concepteurs de la base

Ce sont les personnes qui s'occupent de traduire le modèle précédent en un modèle logique exploitable par le SGBD. Le concepteur est un spécialiste des bases de données qui prépare les tables, les vues, les schémas d'accès. C'est lui qui renseigne les utilisateurs et programmeurs pour la définition des requêtes. Il n'a pas, en principe, à être spécialisé sur un SGBD particulier, mais en pratique les éléments qu'il manipule sont liés au SGBD qui sera employé. C'est ordinairement lui qui crée les éléments nécessaires à la base de données (tables, vues…) en collaboration avec l'administrateur de la base. C'est parfois la même personne qui est en charge de la partie analyse et de la conception, ce qui peut induire une vision un peu trop orientée techniquement – comme celle d'un programmeur qui écrirait le cahier des charges d'une application. Par contre, le concepteur peut aussi être administrateur du SGBD, ce qui ne pose pas de problèmes particuliers d'approche.

5.3 Administrateurs de base de données (DBA, DataBase Administrator)

L'administrateur a la responsabilité du fonctionnement général du SGBD. Il crée les ressources (bases, comptes) à la demande. Il donne les droits d'accès et gère les personnes qui accèdent au système. Il vérifie que les ressources sont suffisantes (taille du disque, puissance de la machine), effectue les sauvegardes, vérifie les failles de sécurité. Pour ces opérations, il est en relation avec l'administrateur système et réseau de la structure. Ce métier est extrêmement lié au SGBD employé. Il n'y a pas vraiment de normalisation pour les opérations d'administration des SGBD qui sont spécifiques au SGBD et à la version utilisés.

5.4 Utilisateurs standard et programmeurs d'applications

Ce sont eux qui utilisent le système d'information. Ils y ont accès grâce aux vues définies par le concepteur de la base. Ils utilisent les schémas déterminés aux deux premières étapes de la conception. Ils n'ont pas besoin théoriquement d'être spécialisés sur le SGBD employé. En pratique il est préférable, surtout pour les développeurs d'applications, d'avoir de bonnes connaissances du fonctionnement du SGBD. Par exemple, pour optimiser les performances, la manière d'écrire les requêtes peut être assez différente suivant le SGBD employé.

6 Plan de l'ouvrage

Le plan de l'ouvrage est déterminé par les différentes étapes de la conception d'une BD.

Le **deuxième chapitre** traite de l'étape d'analyse du monde réel pour en concevoir un modèle descriptif. La modélisation est effectuée classiquement par le modèle « entité-association ». On aborde également la représentation du modèle avec UML.

Le **troisième chapitre** est essentiellement consacré au modèle relationnel et aux opérations qui lui sont associées. On s'intéresse ensuite aux méthodes grâce auxquelles il est possible de passer du modèle précédent à un ensemble de tables du modèle relationnel. Puis, on continue par l'étape de normalisation qui permet de mettre en évidence les incohérences du système d'information ainsi créé et de les rectifier. On présente dans ce chapitre les trois premières formes normales et celle de Boyce-Codd. Une autre approche de la création d'une base de données à partir de la « relation universelle » est abordée à la fin du chapitre.

Le **quatrième chapitre** présente le langage SQL. Après avoir exposé la syntaxe générale du langage, il s'attache à la réalisation en SQL des opérations relationnelles vues précédemment. On aborde ensuite la correspondance entre les questions classiques des bases de données en langage parlé et leur représentation avec SQL. La dernière partie du chapitre traite de la partie « description de données » de SQL, qui permet de créer et gérer les tables dans un SGBD.

Le **cinquième chapitre** constitue un exemple complet de réalisation d'une base de données par la pratique. Cet exemple part de l'énoncé du monde réel en langage parlé jusqu'à la réalisation du système d'information, puis bien sûr son utilisation. Ce chapitre reprend les notions vues aux chapitres précédents de manière pratique.

Le **sixième chapitre** aborde les mécanismes généraux de préservation des données associés aux SGBD. On aborde la sécurité des accès et les sauvegardes, mais aussi les outils qui participent à la sécurisation des données comme les transactions ou les « triggers ».

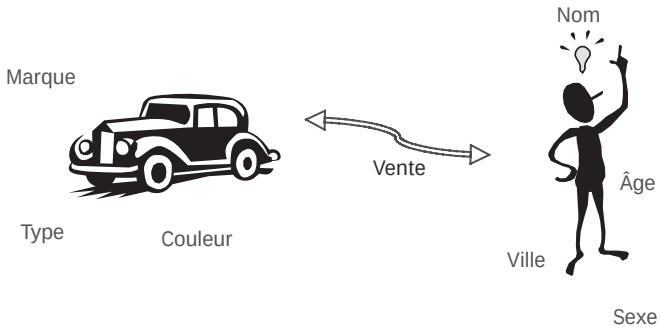
7 Présentation de la BD exemple

Une base de données extrêmement simplifiée est utilisée régulièrement tout au long de l'ouvrage afin d'illustrer les concepts développés au fil de l'ouvrage**.** D'autres bases de données plus complexes sont présentées au fur et à mesure des explications (voir figure 1.14).

Ce système d'information qui constitue l'étude de cas basique modélise l'activité de vente de voitures d'occasion. Dans ce système, deux entités du monde réel sont identifiées : les personnes et les voitures. Une voiture est caractérisée par sa marque, son type, sa couleur. Une personne est caractérisée par son nom, son âge, sa ville, son sexe. L'action modélisée est la vente qui est caractérisée par le prix de la vente et sa date. Une personne peut acheter une plusieurs voitures. Une voiture peut être vendue une seule fois ou jamais.

Figure 1.14

Base de données exemple.



Le modèle « entité-association » et la représentation UML correspondant à cette description seront créés au chapitre 2, « Analyse du monde réel ». Le modèle relationnel sera créé au chapitre 3, « Approche relationnelle ». La base de données résultante sera utilisée pour les exemples du chapitre 4, « SQL ».

Résumé

Une base de données désigne l'ensemble des données stockées. Pour manipuler ces données, on utilise un SGBD (*Système de Gestion de Bases de Données*) qui est un logiciel complexe. L'ensemble composé par les données et le SGBD constitue un système d'information. La conception d'une base de données – de la modélisation du monde réel à son implémentation dans le SGBD – fait appel à des techniques et des méthodes très différentes pour chaque étape. Des métiers spécifiques se sont donc développés autour de ces concepts et les mettent en œuvre. Par exemple, l'approche du monde réel s'apparente à l'analyse faite par un cabinet de consulting alors que l'implémentation dans le SGBD et son administration sont proches des métiers informatiques.

Les SGBD ont évolué parallèlement aux concepts de modélisation des bases de données. On est passé d'une organisation comparable à celle des fichiers informatiques (modèles hiérarchiques ou réseaux) à un modèle plus abstrait : le modèle relationnel. Ce modèle est toujours le plus utilisé actuellement. Il est associé étroitement à SQL, un langage normalisé de description, de manipulation et d'interrogation de données. La modélisation objet, adaptée aux bases de données, n'a pas connu un développement considérable, et ce, malgré les avantages qu'elle procure par rapport au modèle relationnel – en particulier pour le typage des données. Comme c'est le cas dans le domaine de la programmation, une approche mixte semble prendre de l'ampleur : le modèle relationnel-objet. Il s'agit d'apporter au modèle relationnel les possibilités étendues de modélisation procurées par les objets sans remettre profondément en question l'existant.

Le développement des réseaux apporte d'autres manières d'utiliser les bases de données, comme la répartition des données pour améliorer leur disponibilité et leur sécurité. L'interfaçage avec le *World Wide Web* a introduit la prise en compte du langage XML comme format d'échange et de stockage par les SGBD. De nouvelles formes d'interrogation, telles que la « fouille de données » (ou data mining) et les bases de données déductives, permettent d'extrapoler de l'information non explicitement stockée dans les bases de données. Ces approches ainsi que la prise en compte des données multimédias vont faire évoluer les modèles de bases de données et les SGBD que l'on utilise actuellement. Cela se fera probablement sans remettre totalement en cause le modèle relationnel, mais plutôt en le faisant évoluer progressivement.

Exercices

Exercice 1 Notion de base de données

Énoncé

Quelles sont les différences majeures entre un fichier informatique et une base de données gérée par un SGBD ?

Solution

On peut lister ces points essentiels qui les différencient :

- Il n'est pas nécessaire de connaître la méthode de stockage des informations sur le disque pour manipuler les données avec une base de données.
- Un fichier informatique simple n'est pas conçu pour effectuer une recherche d'information par le contenu : pour retrouver le(s) enregistrement(s), on est obligé de parcourir tout le fichier.
- Les modifications de structure (ajout/suppression d'un champ ou modification de sa taille…) nécessitent de recréer un autre fichier et d'y recopier les données.
- Une base de données contient en général plusieurs fichiers dont les enregistrements sont reliés entre eux.

Exercice 2 Recherche dichotomique

Énoncé

Donnez un algorithme intuitif simple de recherche dichotomique en utilisant une table d'index et une table à accès direct.

Solution

Soit V la valeur recherchée, Ti le tableau d'index de taille n. On suppose que la table d'index contient les valeurs du champ indexées dans sa colonne 1 et les numéros d'enregistrement correspondants dans sa colonne 2.

Si le tableau est réduit à un élément dont la valeur dans la première colonne Ti[1,1] est différente de la valeur recherchée alors la recherche est un échec sinon comparer l'élément du milieu z du tableau Ti avec la valeur V

Si l'élément Ti[z,1] est égal à la valeur, accéder à l'enregistrement directement par son numéro Ti[z,2]

Si l'élément Ti[z,1] est inférieur à la valeur recommencer avec la partie basse du tableau (de 1 à z-1)

Si l'élément Ti[z,1] est supérieur à la valeur recommencer avec la partie basse du tableau (de z+1 à n)

EXERCICE 3 LANGAGES D'UN SGBD

Énoncé

On veut supprimer tous les enregistrements qui contiennent la valeur 666 dans le champ 'catégorie'. Utilise-t-on le langage de description de données ou le langage de manipulation de données ? Que se passerait-il si l'on voulait augmenter la taille du champ 'catégorie'.

Solution

Le langage de description de données s'intéresse à la modification de structure d'une table déjà créée ou à la gestion des tables (création/modification). Dans notre cas, on ne touche pas à la structure, on supprime des enregistrements, donc des données de la table. On ne touche pas au dictionnaire de données. On utilisera par conséquent le langage de manipulation de données du SGBD. Pour augmenter la taille du champ, on modifie cette fois la structure même de la table, on utilise alors le langage de description de données.

EXERCICE 4 MODÈLES DE REPRÉSENTATION

Énoncé

Vous devez représenter l'organisation de données correspondant à une classification scientifique d'espèces d'oiseaux. Quel modèle de données (hiérarchique, réseau, relationnel, objet…) choisiriez-vous ?

Solution

Par nature, ce type de données est structuré strictement de manière arborescente et cette structure reste assez stable dans le temps. Il est donc tout à fait possible d'utiliser un simple modèle hiérarchique. Un modèle réseau ne sera pas utile en principe du fait de la structure arborescente des données. On peut également utiliser les modèles relationnel ou objet, mais il n'apporteront pas d'avantage décisif dans ce cas (très) particulier.

EXERCICE 5 MÉTIERS DES BASES DE DONNÉES

Énoncé

Est-il possible de faire réaliser toutes les étapes de la conception d'une base de données par une même personne ? Si oui, quelles sont alors ses compétences minimales ?

Solution

Dans une petite structure, c'est souvent la même personne qui réalise l'ensemble du processus de construction d'une base de données. Ce n'est évidemment pas la bonne méthode, car la vision d'un système d'information élaboré par un administrateur de base de données est très orientée par le SGBD qu'il emploiera. On peut facilement faire le parallèle avec le développement de logiciels où un programmeur va avoir une approche déformée par les préoccupations liées au langage plutôt que d'adopter un point de vue sur la structure générale de l'application. Au minimum, la personne devra disposer des compétences en conception de base de données et en administration du SGBD qui sera utilisé.

EXERCICE 6 UTILISATEURS D'UNE BASE DE DONNÉES

Énoncé

On utilise pour cet exercice la base de données exemple de vente de voitures. On considère les opérations suivantes :

1. Ajouter une personne dans le fichier client.
2. Modifier les possibilités d'ajout dans le champ 'couleur' .
3. Augmenter la taille du champ 'couleur'.
4. Sortir le chiffre d'affaires par marques pour le mois en cours afin de l'importer dans un tableur.
5. Enregistrer une vente.

Quels types d'utilisateurs sont concernés par ces opérations ?

Solution

1. Un utilisateur final (par exemple un vendeur) muni des droits appropriés peut intervenir sur le contenu des données.
2. Pour cette opération, cela dépend s'il s'agit d'une convention ou si cela est entré au niveau des contraintes du SGBD. Dans le premier cas, un utilisateur final peut s'en charger ; dans le second, il faut recourir au concepteur ou à l'administrateur de base de données.
3. Pas d'ambiguïté ici, car on touche à la structure même des données ; cela est du ressort du concepteur ou de l'administrateur de base de données.
4. Il s'agit du domaine du programmeur d'application qui récupère les données en utilisant le SGBD et qui les traite dans un programme pour leur donner leur forme finale.
5. Un utilisateur final (par exemple un vendeur ou la comptabilité) muni des droits appropriés peut intervenir sur le contenu des données.

EXERCICE 7 VUES EXTERNES

Énoncé

On utilise également pour cet exercice la base de données exemple de vente de voitures. On considère trois types d'utilisateurs de la base :

1. les clients ;
2. les vendeurs ;
3. le service comptabilité.

Quelles sont les vues à prévoir pour ces catégories d'utilisateurs ?

Solution

1. Les clients ne doivent avoir accès en lecture qu'aux informations concernant les voitures en stock (non encore vendues).
2. Les vendeurs, s'ils gèrent également le parc de voitures comme c'est souvent le cas, peuvent avoir accès en lecture et en écriture à toutes les données (ventes, voitures, personnes).
3. Le service comptabilité peut avoir accès en lecture à toutes les informations, mais ne peut modifier les informations concernant les voitures ou les personnes du fichier client.

EXERCICE 8 BASE DE DONNÉES RÉPARTIES

Énoncé

On décide de recopier régulièrement une base de données complète sur chacun des six sites de l'entreprise. Quel est l'intérêt de cette solution ?

Solution

C'est une solution coûteuse en ressources, en particulier pour la synchronisation de toutes les mises à jour, mais qui peut remplir deux fonctions :

1. De toute évidence, l'idée est que les différents sites de l'entreprise accèdent à leur copie locale des données. Cela permet d'accélérer les accès et de répartir la charge sur les serveurs locaux de chaque site. Accessoirement, la circulation des requêtes d'interrogation et de mise à jour peut être limitée au réseau local, ce qui renforce la sécurité.
2. Il existe six copies des données sur des sites géographiquement séparés. En cas de sinistre, on peut repartir sans problèmes avec une des copies de la base.

Avant de mettre en place un tel dispositif, on doit se poser la question de la mise à jour des données. Est-ce que les modifications se font uniquement sur la base « maître », ce qui semble plus raisonnable, ou peut-on les effectuer sur toutes les bases et « consolider » ensuite ?

EXERCICE 9 XML

Énoncé

Pourquoi préfère-t-on utiliser XML plutôt que HTML pour représenter les données provenant d'une base de données ? Les données XML sont dites « autodescriptives ». Qu'est-ce que cela signifie et par quel(s) dispositif(s) est-ce réalisé ?

Solution

Par nature, les données contenues dans une base de données sont structurées. Le langage HTML a été conçu pour décrire la mise en forme d'un texte sans considération de sa structure interne. Donc, il n'est pas adapté si l'on désire conserver la structuration des données. Le langage XML a été précisément créé pour décrire la structure des données. Il est toujours possible de passer ensuite du langage XML au langage HTML par une feuille de style ; l'inverse n'est pas possible.

XML permet de représenter des structures de données différentes sous forme arborescente ; il est donc nécessaire de posséder une description de la « grammaire » de la structure. Ce document accompagnateur d'un fichier XML est une DTD, comme pour les fichiers XML classiques, ou plus commodément un schéma XML. Un des avantages du schéma est que l'on peut utiliser les mêmes algorithmes de parcours que pour le fichier XML.

EXERCICE 10 FOUILLE DE DONNÉES ET ENTREPÔTS DE DONNÉES

Énoncé

Par des méthodes d'analyse d'associations, on découvre qu'en utilisant votre système d'information les clients dont le nom commence par M achètent le samedi plus de produits que les autres. Ils génèrent donc un chiffre d'affaires plus important, mais ces produits sont à marge faible et le bénéfice est moins important que pour ceux dont le nom commence par un Z. Qu'avez-vous intégré comme données dans votre entrepôt de données pour pouvoir effectuer cette opération en supposant que votre entreprise soit organisée avec une structure de services classique ?

Solution

Pour obtenir les éléments nécessaires à l'analyse, il nous faut intégrer les données de différents services de l'entreprise.

- Les données du fichier clientèle sont gérées par le service commercial (peut-être avec un tableur ou un traitement de texte pour faire des mailings) afin d'obtenir le nom des clients.
- Les données du service comptabilité permettent d'obtenir les journaux de ventes ; la gestion est faite par exemple par une application de gestion spécifique connectée aux caisses.
- Les données du service achat sur les négociations avec les fournisseurs peuvent être gérées par exemple par une application développée en interne. La marge provenant de la négociation avec les fournisseurs est fluctuante pour le même article au cours du temps en fonction du marché, des personnes qui négocient, etc. Il serait donc intéressant pour ces données de disposer des valeurs « historisées » par périodes pour affiner l'analyse par des tendances.

Pour intégrer ces données provenant de sources différentes dans votre entrepôt de données, vous utiliserez une (ou plusieurs) application(s) de type ETL (*Extract, Transform and Load*) que l'on appelle aussi « datapumping ».

Chapitre 2

Analyse du monde réel

Ce chapitre présente la première étape du processus de modélisation du monde réel, qui consiste à recueillir les informations puis à les transcrire sous une forme conduisant à un passage aisé au modèle relationnel.

On utilise à cette fin le modèle entité-association, dont les concepts et la mise en œuvre sont présentés dans ce chapitre. Cette démarche de modélisation est utilisée depuis plus de vingt ans et présente l'intérêt de proposer une méthode d'analyse simple et efficace dans la majorité des cas.

Au cours des dernières années, la représentation utilisant le formalisme du modèle entité-association est progressivement remplacée par le langage de modélisation UML. Ce dernier apporte, en plus des avantages de la normalisation, de la disponibilité d'outils graphiques logiciels ainsi que des possibilités étendues de description. Les bases de la notation UML sont abordées à la fin de ce chapitre.

1 Démarche d'analyse

1.1 Approche du monde réel

Comment appréhender et simplifier le monde réel, par nature complexe, pour réaliser une modélisation ?

Cette tâche relève de compétences multiples du domaine d'un cabinet de consulting. Il est nécessaire d'identifier les besoins des utilisateurs ainsi que les objectifs et les processus d'alimentation en données des systèmes d'information à concevoir. Les bases de données sont dorénavant au cœur de la plupart des applications, et leur structure doit correspondre au mieux aux attentes de l'organisation.

Son élaboration nécessite différentes étapes et se déroule souvent en même temps que le processus d'**analyse du problème**. Des allers-retours entre ces différentes étapes de la conception sont souvent nécessaires à la constitution du **modèle conceptuel** de la base. On qualifie alors ce processus d'itératif. Il s'agit de construire la structure de la base de données par raffinements successifs.

La première phase de l'analyse du monde réel du problème est réalisée par des **entretiens**, généralement codifiés, avec les utilisateurs. On effectue une analyse du discours pour en extraire l'information utile que l'on resitue dans le contexte de l'organisation en général. L'objectif principal est de guider l'**analyste**, on utilise des **méthodes d'analyse et de conception** issues de l'étude des flux d'information de l'entreprise. Parmi celles-ci, on peut citer la méthode **Merise** d'origine française – très répandue en France dans les années 1980 – ainsi que d'autres issues de la recherche et du génie logiciel, ou spécifiques à des grandes entreprises de consulting.

La présentation de ces méthodes fort complexes dépasse largement le cadre de cet ouvrage. L'objectif de cette section est de donner quelques pistes pour approcher la réalité à modéliser. L'expression des besoins repose sur la formulation du problème à l'aide de phrases simples qui décrivent la réalité à modéliser. Ces phrases se présentent sous la forme « sujet-verbe-complément », avec une tournure active quand cela est possible. Le but est d'obtenir deux types de phrases :

- Celles qui décrivent les liens entre les objets du monde réel – généralement une action ou une propriété. Exemple : *Un lecteur emprunte un livre. Un livre a un auteur.*
- Celles qui caractérisent la manière dont sont reliés ces objets. Exemple : *Un lecteur est considéré comme lecteur s'il a au moins déjà emprunté un livre. Un livre peut être emprunté par plusieurs lecteurs. Il n'y a pas de livres anonymes, un livre est écrit par au moins un auteur.*

On doit ensuite préciser les données qui constituent les objets ainsi que celles qui caractérisent les liens entre les objets.

Remarque

Le terme d'objet du monde réel employé ici n'est pas pris au sens de la programmation objet. Il s'agit plutôt de caractériser un regroupement logique de données. Un titre, un auteur, un éditeur constituent un livre. Un nom, un prénom, un numéro de Sécurité sociale constituent une personne.

1.2 Mise en œuvre

Comment procéder intuitivement pour obtenir ces phrases ?

Un bon point de départ consiste à faire l'inventaire des objets tangibles ou perçus du monde réel. Une fois ces objets identifiés, on cherche à exprimer le (ou les) lien(s) qui permet(tent) de les associer. Par exemple, si l'on doit modéliser une activité de location de DVD, les objets que l'on peut appréhender immédiatement sont les DVD et les clients. En ce qui concerne les liens entre ces objets, on note qu'un client *réserve* un DVD ou qu'un client *loue* un DVD, etc. Ensuite, il faut identifier les objets moins faciles à percevoir directement : les fournisseurs, les acteurs, les réalisateurs… Enfin, une fois les objets identifiés, on cherche à qualifier les liens trouvés. Il faut tenir compte du fait que le lien est toujours à double sens. Par exemple, un client *emprunte* plusieurs DVD. Un DVD *est emprunté* plusieurs fois ou *n'est jamais emprunté* par un client.

Pour obtenir ce résultat, quelles questions faut-il se poser et quelles questions doit-on poser aux acteurs de l'organisation ?

- Décrivez l'activité globalement, en termes simples, sans entrer dans les détails, pour identifier les objets et leurs liens éventuels.
- Indiquez quelles sont les « procédures » utilisées dans l'activité pour caractériser les liens entre les objets. Les procédures permettent d'énoncer les contraintes qui seront intégrées ensuite dans la base de données.

On note que l'on modélise souvent des actions qui représentent une activité, plus rarement des éléments statiques. Les actions représentent fréquemment le lien entre les objets : une personne *emprunte* un DVD, une voiture *est achetée* par un client, etc.

1.3 Notion de temps

Le temps est une notion importante, une base de données modélise des actions qui ont lieu durant une période de temps. Il faut toujours avoir à l'esprit cet aspect pour éviter des erreurs de conception. Une erreur classique est de confondre l'aspect simultané d'une action avec la possibilité de la réitérer durant la période concernée. Lorsque l'on spécifie qu'« un livre peut être emprunté plusieurs fois », il est évident qu'un livre ne peut être emprunté par deux personnes simultanément, mais plutôt qu'il pourra être emprunté à plusieurs reprises durant la période modélisée du fonctionnement de la bibliothèque.

1.4 Cas pratique

Afin d'illustrer les recommandations précédentes, on considère la modélisation très schématique du fonctionnement d'un hôtel. Quelle(s) phrase(s) simple(s) décri(ven)t l'activité de l'hôtel ?

On peut proposer en première approche cette phrase : « Un hôtel loue des chambres à des clients qui effectuent des réservations. » Après avoir procédé à l'analyse de la phrase pour en extraire les parties importantes, on peut la réécrire de la manière suivante :

Un client loue une chambre ; un client réserve une chambre.

Les deux **objets** du monde réel – la chambre et le client – apparaissent clairement. On a identifié ici un double lien entre ces deux objets, cas assez fréquent.

- Ensuite, on s'intéresse à la caractérisation des **liens** : Une chambre peut n'avoir jamais été louée ni réservée.
- Un client est inséré dans le système d'informations à partir du moment où il a effectué soit une réservation soit une location.
- Un client peut réserver ou louer plusieurs chambres.
- Une chambre peut être réservée ou louée plusieurs fois, mais pas pendant la même période de temps.

On rappelle que l'on considère toujours une modélisation associée à une période de temps donnée. Au début du processus, une chambre peut ne pas encore avoir été louée.

Enfin, on décrit les **données** des objets et des liens.

- Un *client* est caractérisé par son *nom*, son *adresse* et son *numéro* de téléphone.
- Une *chambre* est caractérisée par son *numéro*, un *nombre de places*, son *tarif* journalier et la présence ou non d'un *cabinet de toilettes*.
- Une location est caractérisée par une date de début, un nombre de jours et les consommations annexes (petits déjeuners, téléphone…).
- Une *réservation* est caractérisée par une *date de début*, un *nombre de jours* et le versement d'une *avance* éventuelle.

2 Modélisation par le modèle entité-association

Après l'étape de recueil d'informations, on dispose d'un ensemble de phrases simples qui expriment les besoins décrivant la réalité à modéliser : on doit alors en effectuer une représentation. Cette dernière est très importante, car elle est la seule qui donnera une vue d'ensemble des données et des liens qui les caractérisent. Le modèle obtenu à cette étape est en général nommé **Modèle Conceptuel des Données (MCD)**. En effet, on verra lors de l'étape du passage au modèle relationnel, appelé également **modèle logique**, que cette information n'apparaît plus. Les données stockées dans le SGBD seront presque inutiles si l'on ne dispose pas du modèle conceptuel qui est l'équivalent du schéma technique d'un appareil ou du plan d'un bâtiment.

Le formalisme le plus répandu pour constituer ce schéma est le **modèle entité-association** (ou *entity-relationship* en anglais). Il a été présenté à l'origine par P. Chen en 1976 aux États-Unis quasi simultanément avec le modèle de H. Tardieu en France, ce qui explique les notations légèrement différentes en Europe et aux États-Unis, en particulier au niveau de la représentation des cardinalités. Le modèle entité-association a été normalisé à l'ISO. On verra dans la section suivante que l'on peut utiliser également le formalisme **UML** pour la représentation.

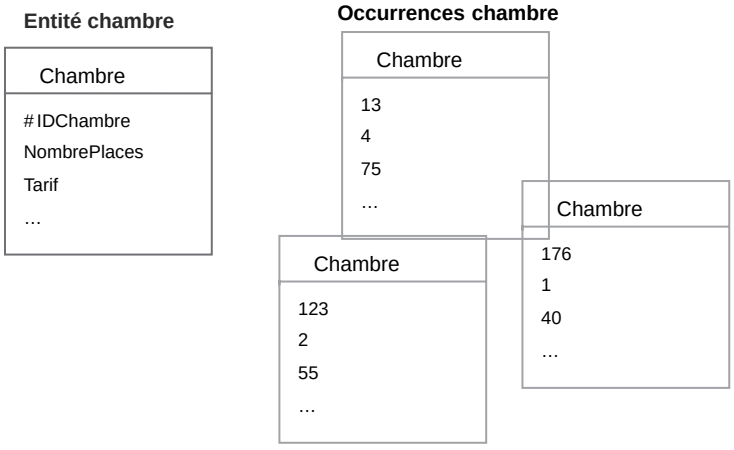
Le formalisme « entité-association », tout comme UML, utilise une représentation graphique sous forme de diagrammes. Les entités sont les objets concrets ou abstraits du monde réel évoqués plus haut. Les associations représentent le lien entre ces entités. Comme on l'a vu précédemment, on peut identifier les entités et les associations en effectuant une analyse du discours, c'est-à-dire des phrases de type « sujet-verbe-complément ». Les sujets et les compléments sont les entités, et le verbe modélise l'association.

2.1 ENTITÉS

Les **entités** sont composées de champs de données que l'on nomme **attributs**. Un attribut, ou un ensemble d'attributs, doit être choisi comme **identifiant** de l'entité, avec pour objectif d'identifier une occurrence (ou représentant) de cette entité. La notion d'identifiant a les mêmes propriétés que la clé dans une relation qui sera introduite au chapitre 3. On représente une entité par un rectangle qui contient le nom de l'entité et ses attributs. L'identifiant est souligné ou précédé d'un caractère '#' (voir figure 2.1).

Figure 2.1

Entités 'chambre', 'attributs' et 'occurrences'.

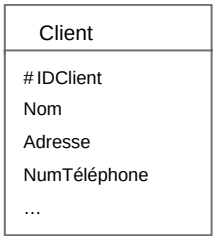


Le choix de l'identifiant n'est pas toujours trivial. Il est parfois nécessaire d'introduire artificiellement un attribut supplémentaire afin de pouvoir disposer d'un identifiant. Dans le cas de l'hôtel, il faudrait intégrer un attribut pour identifier un client (voir figure 2.2), dans la mesure où aucun des attributs issus de l'analyse ne permet d'identifier de manière unique un client. Classiquement, une identification sans ambiguïté reposera sur un numéro unique ; dans notre exemple, c'est l'attribut IDClient.

Les identifiants peuvent être composés par la juxtaposition de différents attributs. Par exemple, on peut identifier un client en juxtaposant les attributs nom+prénom+date_naissance+ville_naissance. En effet, il est peu probable que deux homonymes soient nés le même jour dans la même ville. Cependant, dans la pratique, il est recommandé, autant que faire se peut, de choisir un seul attribut comme identifiant. En effet, il sera plus difficile de vérifier qu'un identifiant composite reste valide lorsque les données évoluent. C'est pourquoi, quand cela est possible, il est indispensable de choisir un identifiant dont les contenus ne sont pas susceptibles d'évoluer au fil du temps. On préfère identifier une personne par un numéro de sécurité sociale que par un numéro de passeport qui a une durée de validité limitée.

Figure 2.2

Entité 'client'.



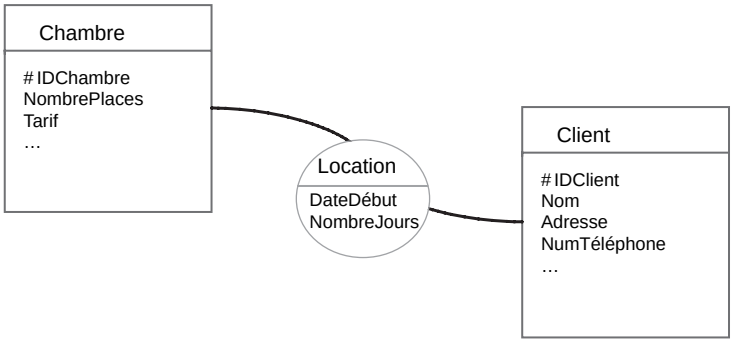
2.2 ASSOCIATIONS

Les **associations** représentent les liens qui existent entre les entités. Elles sont composées le cas échéant d'attributs, bien que cela ne soit pas indispensable. Par conséquent, il n'est pas nécessaire de disposer d'un identifiant pour une association. Lorsque les entités sont associées par deux, elles sont qualifiées de **binaires**. Cependant, il est possible d'en associer plus de deux ; les associations sont alors non plus **binaires**, mais **n-aires**. Le nombre d'entités associées s'appelle le **degré** de l'association.

On représente une association par un ovale qui contient le nom de l'association et ses attributs.

Figure 2.3

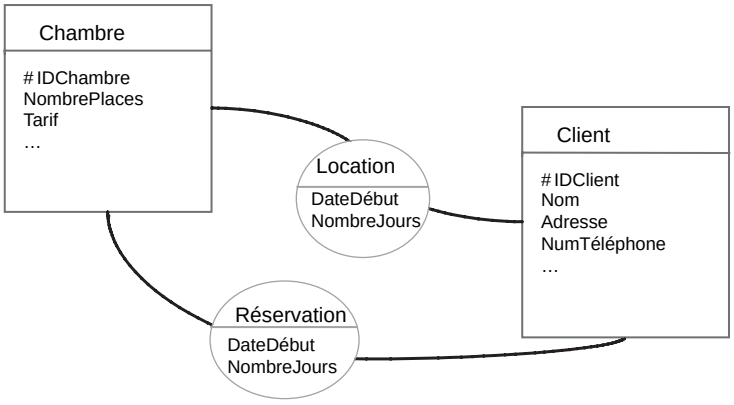
Entités 'client' et 'chambre 'reliées par l'association 'location'.



Il peut également y avoir plus d'une association entre deux entités ; c'est le cas de l'exemple de l'hôtel (voir figure 2.4). Les entités 'client' et 'chambre' sont reliées par deux associations ayant des attributs différents : 'location' et 'réservation'.

Figure 2.4

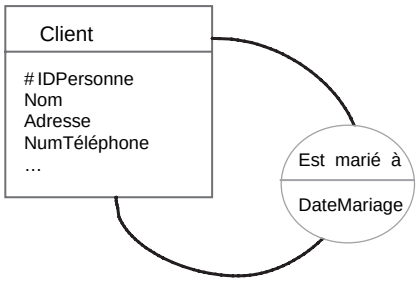
Entités 'client' et 'chambre' reliées par les associations 'location' et 'réservation'.



Enfin, il est possible de relier par une association une entité à elle-même. Si l'on prend l'exemple de la modélisation des liens de mariage entre personnes, on obtient une seule entité 'personne' qui est associée à elle-même par l'association 'est_marié_à' (voir figure 2.5). Dans ce cas, on dit que **l'association est réflexive**.

Figure 2.5

Entité 'personne' reliée par l'association 'est_marié_à'.

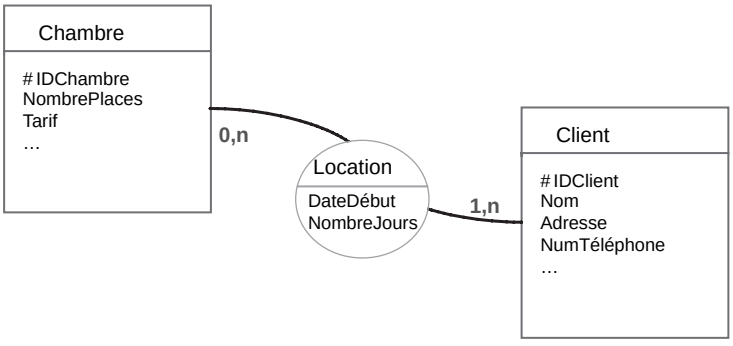


2.3 CARDINALITÉS

Les **cardinalités** décrivent les caractéristiques de l'association entre les entités. On utilise deux nombres qui représentent les valeurs minimales et maximales pour caractériser l'association. Ces nombres modélisent le nombre d'occurrences minimales et maximales des entités impliquées dans l'association. Par exemple, comme illustrée sur la figure ci-après (voir figure 2.6) , une association binaire sera caractérisée entièrement par quatre nombres. En effet, chaque entité participe de manière différente à l'association.

Figure 2.6

Cardinalités d'une association binaire.



Les cardinalités peuvent prendre les valeurs suivantes :

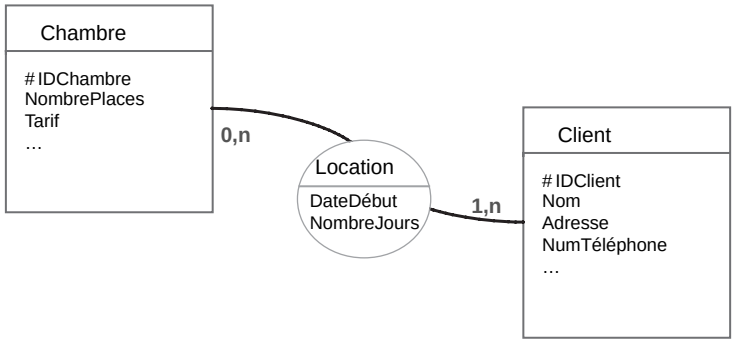
- De un à un, notée **1,1**. Une brosse à dents possède en théorie un (1) et un (1) seul propriétaire.
- De un à plusieurs, notée **1,n**. Un livre a au moins un (1) auteur ; il peut en posséder plusieurs (n). On ne considère pas les ouvrages anonymes.
- Optionnel, notée **0,1**. Une personne est célibataire (0) ou mariée (légalement…) à une (1) autre personne au plus.
- De zéro à plusieurs, notée **0,n**. Un appartement peut être libre (0) ou habité éventuellement par plusieurs habitants (n).

Dans l'exemple de l'hôtel précédent, l'analyse préalable permet de déduire les propriétés suivantes (voir figure 2.7) :

- Un client loue au minimum une (1) chambre ; il peut en louer plusieurs (n). La cardinalité est donc de 1,n.
- Une chambre peut être louée plusieurs fois (n) et elle peut ne pas être occupée (0). La cardinalité est donc de 0,n.

Figure 2.7

Entités 'client', 'chambre' et association 'location' avec cardinalité.



Remarque

Les cardinalités se notent différemment dans le modèle de Chen employé aux États-Unis et dans celui utilisé en Europe. Dans le modèle européen, on dispose les cardinalités du côté de l'entité concernée. La cardinalité '0,n' déduite de « une chambre peut être louée plusieurs fois et elle peut ne pas être occupée » se trouve du côté de l'entité 'chambre'. Cette notation présente l'avantage d'être plus cohérente lors de l'utilisation d'associations « n-aires ». Il n'y a pas de changement dans l'emplacement des cardinalités des entités associées.

2.4 Cas d'étude « casse »

On a présenté dans le chapitre d'introduction la base de données exemple « casse », qui décrit l'activité de vente de voitures d'occasion. Voici sa modélisation sous forme de modèle entité-association. On rappelle la description de ce système.

Deux entités du monde réel sont identifiées : les personnes et les voitures. Une voiture est caractérisée par sa marque, son type, sa couleur. Une personne est caractérisée par son nom, son âge, sa ville, son sexe. L'action modélisée est la vente, caractérisée par le prix de la vente et sa date. Une personne peut acheter plusieurs voitures ou aucune. Une personne peut acheter aucune ou plusieurs voitures. Une voiture peut être vendue ou non.

Recherche des identifiants

Les deux entités 'voiture' et 'clients' ainsi que leurs attributs respectifs sont bien définis ; il manque pour chacune de ces entités un identifiant capable de différencier les occurrences (ou représentants) des entités voitures et des clients. Sur cet exemple simple, il est facile de vérifier qu'aucun attribut ni ensemble d'attributs ne permet de définir sans ambiguïté un identifiant. Le processus de recherche d'un identifiant à partir des relations de dépendances entre les attributs, est détaillé au chapitre 3.

Recherche des cardinalités

Les cardinalités sont déduites des deux dernières phrases :

- **0,n.** Pour « une personne peut acheter plusieurs (n) voitures ou aucune (0) ».
- **0,1.** Pour « une voiture peut être vendue une seule fois (1) ou jamais (0) ».

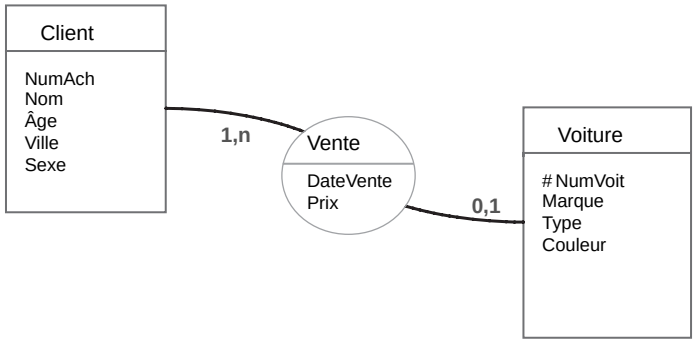


Figure 2.8

Entités 'voiture' et 'client' liées par l'association 'vente' avec cardinalités.

3 Remise en cause et évolution du modèle

Cette section énonce quelques conseils pour améliorer et affiner le modèle conceptuel obtenu à l'étape précédente.

3.1 Qualité des attributs

Dans cette sous-section, on aborde les qualités générales que doivent posséder les attributs des entités et des associations. Des règles simples permettent de guider le choix des attributs.

Une première règle est de ne stocker que les **attributs** strictement **nécessaires** : la tendance des utilisateurs est en général de prévoir le plus d'attributs possible juste au cas où. Le choix s'effectue en tenant compte des fonctionnalités attendues du système d'information par élimination systématique des données non utiles.

Une deuxième règle consiste à ne pas retenir les **attributs** qui **peuvent être déduits** d'autres attributs. La déduction peut se faire soit par le calcul, soit par un lien sémantique entre plusieurs attributs :

- Le chiffre d'affaires peut être calculé à partir du prix de vente et de la quantité.
- Une référence peut être construite à partir du nom du produit et du fournisseur.

Le but est de réduire la place occupée ainsi que les risques d'incohérence.

Enfin, une dernière règle concerne le nom des **attributs** qui doivent être « **parlants** ». Il ne faut pas hésiter à utiliser des noms de grande taille si cela facilite la compréhension du rôle de l'attribut dans l'entité. Par exemple, il est préférable d'utiliser un attribut 'Numéro_Série' plutôt qu'un attribut 'ns'. Dans la mesure du possible, on essaie de donner des noms d'attributs spécifiques pour chacune des entités. Ces deux dernières précautions faciliteront la compréhension générale du modèle et son utilisation future dans le SGBD.

3.2 Réorganisation des entités

Dans cette sous-section, on remet en cause le découpage en entités produits par une première approche. Le processus itératif de cette remise en cause peut conduire à un **redécoupage** des entités ou à leur **fusion**.

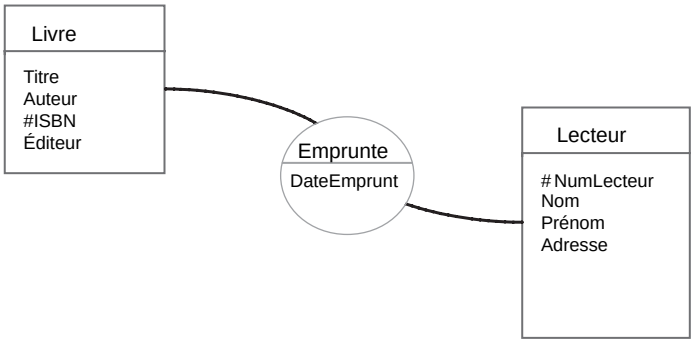
On considère l'exemple de la bibliothèque de prêt, que l'on a déjà utilisé précédemment. La phrase caractéristique décrivant l'activité de la bibliothèque est la suivante : « Un lecteur

emprunte un livre. » On en déduit les deux entités 'lecteur' et 'livre' liées par l'association 'emprunte' (voir figure 2.9).

- Un lecteur est caractérisé classiquement par son nom, son prénom et son adresse. Comme il n'y a pas d'attribut possédant les caractéristiques d'un identifiant pour l'entité 'lecteur', on ajoute un attribut identifiant. Cet attribut supplémentaire est le numéro de lecteur. À noter que dans le monde réel, ce numéro pourrait correspondre par exemple à un numéro de carte de lecteur.
- Un livre est caractérisé par son titre, son auteur, son numéro ISBN, son éditeur. Le numéro ISBN est ici un identifiant pour l'entité puisqu'il est unique.
- L'association 'emprunte' a pour attribut la date d'emprunt.

Figure 2.9

Entités 'livre' et 'lecteur' liées par l'association 'emprunte' (sans cardinalités).



Cet exemple simple permet de se poser quelques questions qui vont conduire à une réorganisation du modèle.

Que se passe-t-il si l'on possède plusieurs exemplaires du même ouvrage ?

Le numéro ISBN n'est plus identifiant puisqu'il est le même pour chacun des exemplaires. Une solution consiste à ajouter un numéro supplémentaire unique pour chaque livre, qui correspond à la notion de cote dans une bibliothèque. Ainsi, même si l'on a dix exemplaires d'un ouvrage, il est possible de les différencier. L'inconvénient de cette solution est que l'on répète les informations communes aux différents ouvrages (titre, auteur…) à chaque exemplaire. L'un des risques est de répéter incorrectement ces informations et d'aboutir ainsi à des incohérences.

La solution correcte dans ce cas est de séparer l'entité 'livre' en deux entités 'livre' et 'ouvrage' (voir figure 2.10). L'activité de la bibliothèque est alors décrite par deux phrases :

- Un lecteur emprunte un exemplaire.
- Un livre représente un exemplaire d'un ouvrage.

Un livre est caractérisé par un numéro (cote) identifiant et un ISBN. Un ouvrage est caractérisé par un ISBN, qui est bien dans ce cas un identifiant, un titre, un auteur et un éditeur. Dans cet exemple, l'objet du modèle conceptuel 'livre', utilisé comme entité, est décomposé en deux autres entités dont une est abstraite : l'ouvrage. L'ouvrage est un regroupement d'attributs qui n'a pas d'existence « tangible » dans le monde réel.

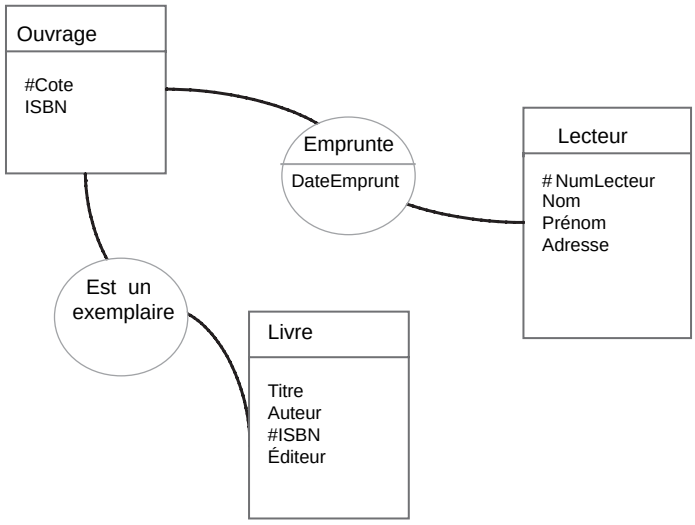


Figure 2.10

Entités 'livre', 'ouvrage' et 'lecteur' liées par les associations 'emprunte' et 'est un exemplaire' (sans cardinalités).

Que se passe-t-il si l'ouvrage possède plusieurs auteurs ?

Une solution simpliste est de prévoir un champ par auteur supplémentaire, c'est-à-dire d'ajouter des champs 'auteur2', 'auteur3', 'auteur4', etc. Cette solution pose de nombreux problèmes :

- Si seulement dix livres sur un million possèdent plusieurs auteurs, on réserve la place pour les champs auteurs supplémentaires qui sera inutilisée.
- Si un livre possède un nombre d'auteurs supérieur au nombre de champs prévus, on ne résout pas le problème.
- Si l'on considère qu'un auteur peut avoir écrit plusieurs ouvrages, on répète dans ce cas les informations le concernant pour chacun de ses ouvrages. Cela constitue un cas typique de redondance qui risque de provoquer des incohérences.

La solution correcte dans ce cas est de créer une entité supplémentaire pour ces attributs qui sont sémantiquement de même type. On créera ici une entité auteur qui contiendra un numéro d'auteur (identifiant), son nom et son prénom. Cette entité est reliée à l'entité ouvrage par l'association 'a_écrit' (voir figure 2.11).

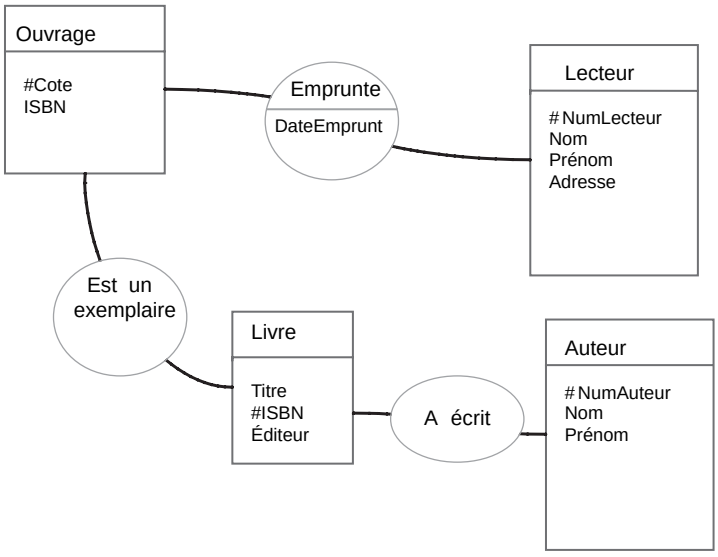


Figure 2.11

Entités 'livre', 'ouvrage', 'auteur' et 'lecteur' liées par les associations 'emprunte', 'a_écrit' et 'est un exemplaire' (sans cardinalités).

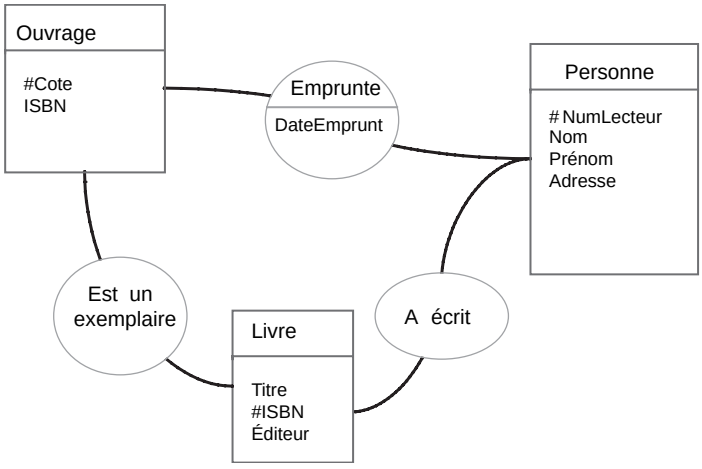
Que se passe-t-il si un auteur emprunte un livre ?

Un auteur peut également emprunter un livre et revêtir par conséquent le rôle de l'emprunteur. Dans ce cas, on répète les informations le concernant dans l'entité 'lecteur'. De même que précédemment, cela provoque de la redondance et peut générer des incohérences.

Comme les entités 'lecteur' et 'auteur' ont la même structure (c'est-à-dire des mêmes attributs), la solution consiste à les fusionner en une entité unique 'personne'. Cette opération conduit à une association entre les entités 'personne' et 'livre' (emprunte) et les entités 'personne' et 'ouvrage' (a_écrit) (voir figure 2.12).

Figure 2.12

Entités 'livre', 'ouvrage', 'personne' liées par les associations 'emprunte', 'a_écrit' et 'est_un_exemplaire' (sans cardinalités).



On a identifié dans cette sous-section plusieurs cas qui nécessitent de réorganiser les entités obtenues lors d'une première analyse :

- Si l'on repère à l'intérieur d'une entité des attributs qui représentent un ensemble logique (mis en valeur dans l'exemple par un défaut d'identifiant pour un livre), on sépare ces attributs dans une entité supplémentaire.
- Si des attributs dans une même entité possèdent une sémantique identique (auteurs, numéros de téléphone multiples), on crée une entité supplémentaire pour séparer ces attributs. Cette entité sera associée à celle dont proviennent les attributs à l'origine.
- Si des entités ont la même structure (et représentent le même type d'objet), on les fusionne et l'on conserve les associations qui existaient avant la fusion.

On retrouvera ce processus de remise en question du modèle à un autre niveau, lors de l'étape de normalisation qui est abordée au chapitre 3.

3.3 Élimination et fusion d'associations

Dans cette sous-section, on s'intéresse à la réorganisation d'associations qui n'ont pas de raison d'être, par une fusion des associations ou par leur intégration dans les entités.

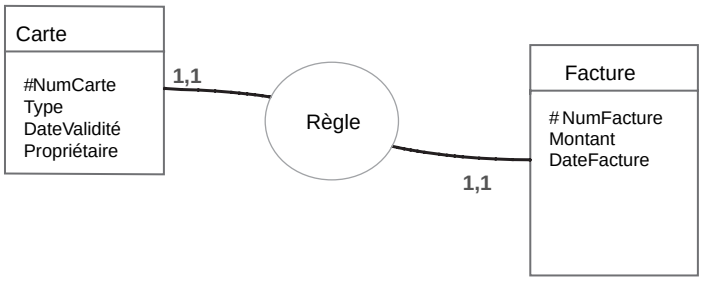
Élimination d'associations

On considère le cas d'un acte d'achat effectué sur Internet avec une carte bancaire. Une facture est réglée par une carte. On peut distinguer les deux entités 'facture' et 'carte' . Une facture est identifiée par un numéro de facture, et est constituée d'un montant et d'une date. Une carte bancaire est identifiée par son numéro, son type (Visa, MasterCard), sa date de validité et son propriétaire. Les cardinalités entre les entités 'carte' et 'facture' sont de type 1-1 (une facture est payée par une et une seule carte) et 1-n (une

carte peut servir à régler plusieurs factures). On peut utiliser une Ecarte qui permet d'améliorer la sécurité de ces transactions. Une Ecarte est une carte « virtuelle » associée à une véritable carte bancaire, valable pour une seule transaction. Les cardinalités deviennent alors de type 1-1 des deux côtés : une Ecarte ne permet de régler qu'une et une seule facture (voir figure 2.13).

Figure 2.13

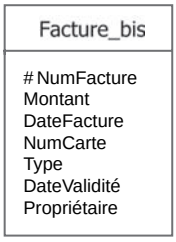
Entités 'carte' et 'facture' liées par l'association 'règle' avec ses cardinalités.



Dans ce cas, l'association 'règle' n'a plus lieu d'être puisqu'il s'agit d'une pure bijection : à une facture correspond une Ecarte et une seule, et à une Ecarte correspond un produit et un seul. On peut fusionner les deux entités 'carte' et 'facture' et éliminer l'association (voir figure 2.14).

Figure 2.14

Entité 'facture_bis' fusionnée.

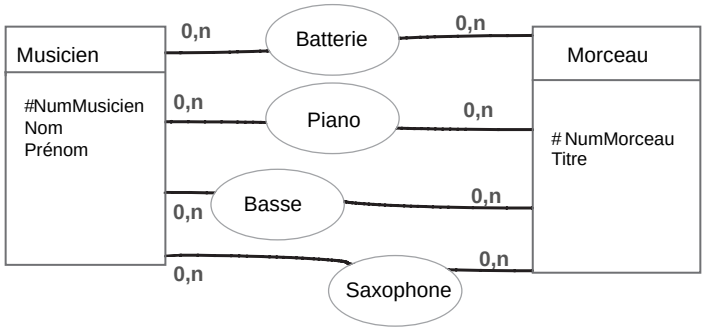


Fusion d'associations

Suivant le même principe, il est possible de fusionner plusieurs associations ayant le même rôle sémantique. Si l'on considère la description de l'activité suivante, liée à l'exécution de morceaux de jazz en quartet. Pour un morceau donné, le premier musicien joue la partie de basse, le deuxième celle de batterie, le troisième celle de piano et le quatrième celle de saxophone (voir figure 2.15).

Figure 2.15

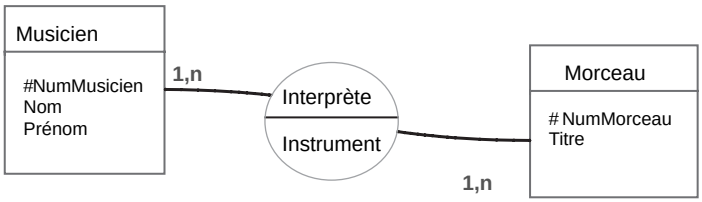
Entités 'musicien' et 'morceau' liées par les associations 'basse', 'batterie', 'piano' et 'saxophone'.



Comme il s'agit de la même activité (un musicien interprète un morceau), on peut remplacer toutes les associations par une association 'interprète'. L'association 'interprète' contiendrait l'attribut 'instrument' pour ne pas perdre l'information de l'instrument associé (voir figure 2.16).

Figure 2.16

Entités 'musicien' et 'morceau' liées par l'association 'interprète'.



Remarque

Il en serait différemment pour des associations entre ces deux entités qui exprimeraient un autre sens. Si l'on considère l'association 'compose' ou 'arrange' entre les entités 'musicien' et 'morceau', il ne s'agit pas de la même activité que l'association 'interprète' : on ne fusionnera pas les associations dans ce cas.

Dans cette section, on a essayé de porter un regard critique sur le modèle conceptuel « brut » issu d'une première phase d'analyse que l'on a obtenue en utilisant les techniques énoncées dans les sections précédentes. Cette étape fait partie intégrante du processus itératif de constitution du modèle par raffinements successifs. Le processus se poursuivra à un autre niveau par la normalisation des relations déduites de ce modèle. Cette partie sera traitée au chapitre 3.

4 Représentation avec UML

Le langage UML (*Unified Modeling Language*) est un standard élaboré à l'initiative de l'OMG (*Object Management Group*). Il est donc spécifiquement destiné à la modélisation objet. Comme son nom l'indique, UML est le résultat d'un processus de fusion (unification) de différentes méthodes existantes dans le domaine de la conception objet : Booch, OMT, OOSE. Ces concepts reposent principalement sur les travaux de Grady Booch, James Rumbaugh et Ivar Jacobson. C'est un langage indépendant de la plate-forme utilisée ainsi que des autres langages de plus bas niveau employés en programmation objet, comme Java et C++. Il utilise une notation graphique et a connu très rapidement le succès dans le monde objet par la souplesse qu'il apporte en gestion de projets de développement.

4.1 UML et bases de données

Quel est le rapport avec les bases de données ? UML a été adapté afin d'être utilisé pour la modélisation en bases de données : on recourt à cet effet à un profil spécifique (*UML profile for data modeling*). Même s'il est prévu initialement uniquement pour la modélisation objet, UML permet aussi de représenter un modèle conceptuel utilisable pour créer des bases de données relationnelles et relationnelles-objet. Les approches objet et relationnelles-objet en bases de données ont été introduites dans le premier chapitre et dépassent largement le cadre de cet ouvrage.

En outre, une motivation supplémentaire pour utiliser UML est de pouvoir disposer d'outils logiciels graphiques du marché pour la description et l'automatisation du processus de passage au SGBD, allant jusqu'à la génération de code SQL. UML offre des garanties de pérennité du fait de sa normalisation. Même si UML n'apporte pas d'évolutions majeures quant aux méthodes existantes de modélisation en bases de données, il est de plus en plus largement répandu et permet d'utiliser le même formalisme et les mêmes outils que les concepteurs d'applications logicielles.

Remarque

UML est un formalisme de représentation. Il n'est associé à aucune méthode particulière comme peut l'être la méthode Merise qui associe une méthodologie et un langage de description.

4.2 Représentation du modèle conceptuel avec les diagrammes de classes UML

UML a été développé à l'origine pour la modélisation des concepts objet. La tendance actuelle consiste à utiliser une partie restreinte de ce puissant langage de modélisation, afin de représenter le modèle conceptuel des données. Dans sa version complète, UML propose neuf types de diagrammes répartis en deux catégories : les diagrammes de **structure** qui décrivent la partie statique du modèle ; les diagrammes de **comportement** qui décrivent la partie dynamique, c'est-à-dire les traitements. On utilise les **diagrammes de classes** qui font partie des diagrammes de structure pour représenter les éléments du modèle conceptuel.

Classes

Dans ce type de représentation les entités sont interprétées comme des **objets** et sont représentées par des **classes**. La description d'une classe en UML comprend trois parties :

- le nom de la classe ;
- la description des attributs ;
- les méthodes associées à l'objet.

Cette section n'utilise pas la notion de méthode spécifique à l'approche objet. La description des attributs peut être réalisée de manière plus précise. UML offre la possibilité de décrire à ce niveau le domaine de l'attribut :

jour : (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche)

Une classe est représentée graphiquement par un rectangle séparé en trois zones correspondant aux trois parties précédentes (voir figure 2.17).

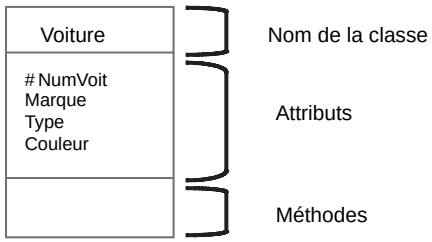


Figure 2.17

Entité 'voiture' représentée par une classe UML.

Associations et multiplicité

Les classes sont reliées par des **associations** identifiées par leur nom. Si l'association possède elle-même des attributs, ils sont intégrés dans une classe d'association. L'expression des cardinalités est quasiment la même en UML que pour le modèle entité-association ; elles sont appelées **multiplicité**. UML permet de définir plus précisément les cardinalités en utilisant les valeurs suivantes.

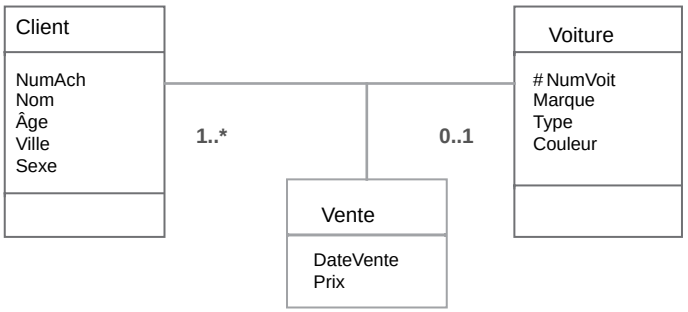
- '1'. Obligatoire, un et un seul (peut s'écrire également 1..1).
- '*'. Non obligatoire (peut s'écrire également 0..n).

- '0..1'. Non obligatoire restreint à 1.
- '1..*'. Obligatoire.
- '2..4'. Obligatoire restreint de 2 à 4.

Les associations sont matérialisées par un trait entre les classes représentant les entités. Le nom de l'association est inscrit au-dessus du trait. Dans le cas où l'association possède des attributs, on utilise une classe d'association sans nom qui contiendra les attributs. La classe d'association est reliée par un trait pointillé (voir figure 2.18).

Figure 2.18

Entités 'voiture' et 'client' liées par l'association 'vente' représentées par une classe UML.



Remarque

Attention, la position des cardinalités dans un diagramme de type UML est inversée par rapport à celle utilisée dans un diagramme de type modèle entité-association présentée dans ce chapitre. En effet, UML utilise la notation employée aux États-Unis, mal adaptée à la représentation d'association de degré supérieur aux associations **binaires**.

On peut décrire plus finement l'association en spécifiant un **rôle** de l'association pour chacune des classes associées. C'est-à-dire que l'on distingue, du point de vue de chacune des classes impliquées dans l'association, la manière dont elle est associée à une autre classe par un terme spécifique : le rôle. Comme les cardinalités sont différentes pour chacune des classes impliquées dans une association, on dispose alors d'une description complémentaire. La notion de rôle est surtout utile lorsqu'une classe est associée à elle-même.

Par exemple, si l'on considère les classes 'voiture' et 'client' (voir figure 2.19), du point de vue de la classe 'voiture', la classe 'client' a le rôle d'acheteur. Du point de vue de la classe 'client', la classe 'voiture' a le rôle d'une dépense. Ainsi, on note le nom du rôle au même emplacement que celui de l'association (au-dessus du trait), mais du côté de la classe concernée.

Figure 2.19

Classes 'voiture' et 'client' liées par l'association 'vente' avec des rôles.

4.3 Du modèle entité-association à UML

La notation UML est utilisée dans cette section pour représenter les entités et les associations issues de la démarche d'analyse « primitive » que l'on a décrite dans la première sec-

tion. La correspondance qui est faite avec celle fondée sur le modèle entité-association est la suivante :

- Une entité est représentée par une classe.
- Une association est représentée par une association.
- Une cardinalité est représentée par une multiplicité.

À ce niveau d'utilisation, les avantages qu'apporte la notation UML n'apparaissent pas clairement. L'intérêt réside dans la possibilité d'utiliser des outils logiciels. Ces derniers, à partir du schéma du modèle conceptuel exprimé en UML, automatisent, ou plutôt assistent, le processus de passage au modèle relationnel qui sera abordé au chapitre 3.

L'idée à terme est de générer automatiquement le langage SQL nécessaire, comme cela se fait en génie logiciel où l'on génère le code du langage utilisé. En revanche, l'utilisation d'UML prendra tout son sens pour une modélisation plus complexe utilisant par exemple le concept objet d'héritage qui dépasse le cadre de cet ouvrage.

Résumé

Ce chapitre aborde l'approche de la modélisation, qui consiste d'abord à extraire les informations du monde réel puis à en proposer une représentation formelle appelée modèle conceptuel. Il n'existe pas réellement de démarche « scientifique » pour réaliser cette étape. Elle est plutôt constituée d'un ensemble de techniques diverses qui permettent d'appréhender la complexité du monde réel à modéliser et de la simplifier.

L'analyse du monde réel a pour but de pouvoir identifier les données qui interviennent dans le domaine considéré et de les regrouper dans des objets. L'étape suivante consiste à caractériser les liens qui les unissent. À cette fin, on décrit le système à modéliser par des phrases simples de type « sujet-verbe-complément », que l'on peut classer en deux grandes catégories :

- celles qui décrivent les objets du monde réel et les liens qui les unissent : le sujet et le complément sont représentés par des **entités** (classes) et le verbe est représenté par une **association ;**
- celles qui décrivent la manière dont sont reliés ces objets : on en déduira les cardinalités (ou multiplicités dans le cas d'utilisation d'UML) des associations.

Les entités sont constituées de champs de données que l'on nomme « attributs ». Pour identifier de manière unique les représentants d'une entité, appelés également « instance », un attribut (ou un ensemble d'attributs) est choisi : on l'appelle « identifiant ». Une fois les éléments à modéliser identifiés, leur représentation normalisée recourt à différents langages. Les deux types de formalismes les plus courants employés sont les suivants :

- **Le modèle entité-association.** Il a fait ses preuves ; sa notation est très intuitive et il est encore couramment utilisé.
- UML (*Unified Modeling Language*). Il représente (probablement) l'avenir, car il est soutenu par une communauté très importante qui dépasse celle des bases de données. Il offre l'avantage d'être bien adapté à la modélisation de données complexes qui nécessitent une approche objet.

Ni le modèle entité-association, ni UML ne sont des méthodes d'analyse ; il s'agit dans les deux cas de simples formalismes de représentation.

Exercices

Exercice 1 Identifiant d'une entité

Énoncé

On considère l'entité ci-après, qui décrit des salles de cinémas. Les attributs de cette entité sont les suivants :

- nom de la salle ;
- nom du cinéma ;
- ville du cinéma ;
- nombre de places ;
- taille de l'écran.

Que proposez-vous comme identifiant pour cette entité ?

Solution

Si l'entité décrit des cinémas situés sur le territoire français par exemple, le nom du cinéma n'est pas significatif : le *Rex* existe dans bon nombre de villes. Il en est de même pour le nom de la salle : on peut imaginer sans peine que plusieurs cinémas possèdent une salle « John Cassavettes ».

Les autres attributs peuvent difficilement prétendre à identifier une salle de cinéma : le nombre de places et la taille sont communs à beaucoup de salles ainsi que la ville. Si l'on essaie la combinaison 'Nom du cinéma & Nom de la salle', on court le risque que deux cinémas *Caméo* disposent de la même salle « François Truffaut ».

Finalement, un identifiant possible serait la combinaison 'Nom du cinéma & Nom de la salle Ville du cinéma'. Il est peu probable en effet que la même ville dispose de plusieurs cinémas du même nom… encore que ce soit du domaine du possible. Dans ce cas, il est préférable de créer un champ identifiant de toutes pièces qui identifiera la salle. Classiquement, on utilisera un chiffre identifiant que l'on peut nommer 'Numero_salle'.

Exercice 2 Identification des entités et des associations

Énoncé

On veut modéliser l'activité de vente de billets pour un théâtre. Quelles phrases vont nous permettre d'identifier les entités et la manière dont elles sont associées ? Proposez les attributs que vous utiliseriez pour décrire ces entités et leurs associations ainsi que les identifiants de chaque entité. Que se passe-t-il si le prix du billet varie pour chaque séance et en fonction de la place ?

Solution

Les éléments qui s'imposent intuitivement sont les spectateurs et la pièce jouée. La phrase qui représente le lien entre ces entités peut être de la forme : « les spectateurs achètent un billet pour une pièce ».

On identifie deux entités 'spectateur' et 'pièce' liées par l'association 'achat'.

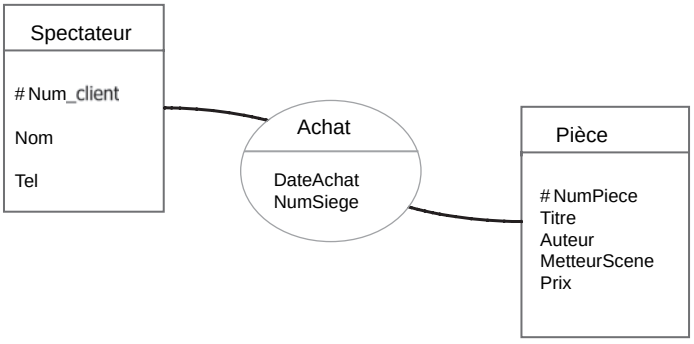
L'entité 'spectateur' contient les attributs nom, numéro de téléphone. On crée un identifiant 'numero_client' puisque 'nom' et 'numéro de téléphone' ne sont pas identifiants. On

pourrait éventuellement utiliser la combinaison des deux attributs comme identifiant, car deux personnes de même nom ont rarement le même numéro de téléphone, mais il est préférable de créer un identifiant dans ce cas.

L'entité 'pièce' comprend un titre, l'auteur, le metteur en scène et le prix d'une place (on suppose que toutes les places sont au même prix). De même que pour l'entité 'spectateur', on a besoin de créer un identifiant pour la pièce.

L'association a pour attributs la date et le numéro de siège. Ces attributs sont en effet caractéristiques de l'action d'achat et non pas du spectateur ou de la pièce (voir figure 2.20).

Figure 2.20
Entités 'spectateur' et 'pièce' liées par l'association 'achat' (sans cardinalités).



Si le prix des places varie pour chaque séance, il est non plus une caractéristique de la pièce, mais de l'achat. L'attribut 'prix' devient un attribut de l'association 'achat'. Il n'y a jamais de solution unique en base de données. On aurait pu, par exemple, utiliser une entité 'billet' liée aux deux entités 'spectateur' et 'pièce'.

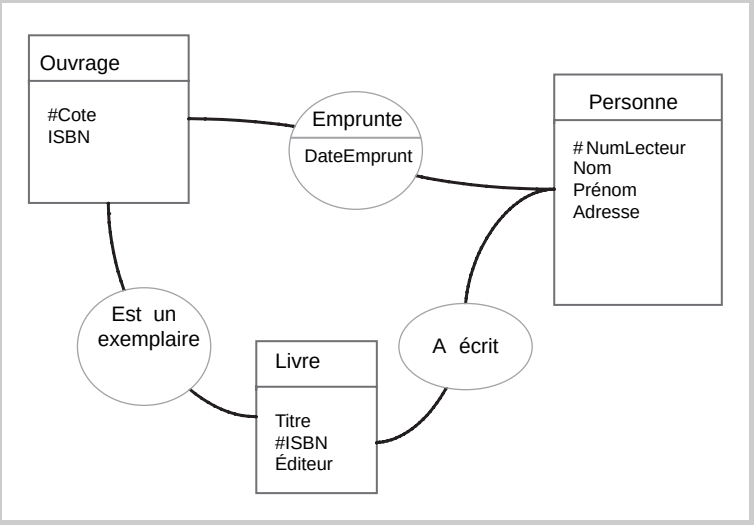
EXERCICE 3 QUESTIONS ASSOCIÉES AUX CARDINALITÉS

Énoncé

Dans l'exemple sur la bibliothèque, présenté précédemment dans ce chapitre, on est parvenu au modèle suivant à la suite de la remise en cause du modèle (voir figure 2.21) :

- Les entités 'livre' 'ouvrage' sont liées par l'association 'est_un_exemplaire'
- Les entités 'personne' et 'livre' sont liées par l'association 'emprunte'.
- Les entités 'personne' et 'livre' sont liées par l'association 'a_écrit'.

Figure 2.21
Entités 'livre', 'ouvrage', 'personne' liées par les associations 'emprunte', 'a_écrit' et 'est_un_exemplaire' (sans cardinalités).



Énoncé (*suite*) Quelles questions faut-il poser aux utilisateurs de la base de données pour déterminer les cardinalités des associations ? Proposez une réponse à ces questions et déduisez-en les cardinalités pour chaque entité.

Solution

On doit déterminer pour chaque association les deux cardinalités minimales et maximales associées aux deux entités liées, puisqu'il s'agit d'associations binaires. Il y aura donc quatre questions à (se) poser par association. On pose ici les questions et on propose les réponses ; dans la vie réelle, ce sont les utilisateurs de la base de données qui apporteraient les réponses (voir figure 2.22).

Association 'est_un_exemplaire'

Pour déterminer les cardinalités du côté de l'entité 'ouvrage :

- Peut-on avoir une fiche d'ouvrage sans disposer du livre lui-même dans la bibliothèque ? Si c'est le cas, la valeur minimale sera de '0', sinon elle sera '1'. La valeur '1' est plus logique dans ce contexte, sauf si l'on a récupéré le catalogue descriptif d'un éditeur.
- Une fiche d'ouvrage peut-elle servir à plusieurs livres ? Si c'est le cas, la valeur maximale sera de n ; sinon, elle sera de '1'. C'est le but ici de regrouper les informations d'ouvrages communes à plusieurs exemplaires. La cardinalité maximale choisie est de 'n'.

Pour déterminer les cardinalités du côté de l'entité 'livre' :

- Peut-on avoir un livre sans avoir de fiche ouvrage ? Si c'est le cas, la valeur minimale sera de '0', ; sinon, elle sera de '1'. Il paraît clair que tout livre a une fiche ouvrage, à moins qu'il ne soit juste identifié dans la bibliothèque mais pas encore catalogué.
- Un livre peut-il avoir plusieurs fiches ouvrage ? Si c'est le cas, la cardinalité maximale sera de 'n' ; sinon, elle sera de '1'. On peut supposer qu'un livre n'a qu'une fiche ouvrage.

Association 'emprunte'

Pour déterminer les cardinalités du côté de l'entité 'livre' :

- Un livre peut-il n'avoir jamais été emprunté ? Si c'est le cas, la valeur minimale sera de '0' ; sinon elle sera de '1'. On choisit '0' : un livre peut ne rencontrer aucun succès.
- Un livre peut-il être emprunté plusieurs fois ? Si c'est le cas, la valeur maximale sera de 'n' ; sinon elle sera de '1'. On rappelle qu'une base de données modélise une activité sur une période de temps. Un livre peut être emprunté plusieurs fois même si l'on ne peut pas le prêter à deux personnes simultanément. On choisit 'n'.

Pour déterminer les cardinalités du côté de l'entité 'personne' :

- Une personne peut-elle ne pas avoir emprunté de livre ? Si c'est le cas, la valeur minimale sera de '0' ; sinon, elle sera de '1'. Une personne ne s'inscrit à la bibliothèque que dans le but d'emprunter un livre ; elle en a donc au moins emprunté un. On choisit '1'.
- Une personne peut-elle emprunter plusieurs livres ? Si c'est le cas, la cardinalité maximale sera de 'n' ; sinon, elle sera de '1'. On choisit 'n' : un lecteur n'est pas limité à l'emprunt d'un seul livre.

Association 'a_écrit'

Pour déterminer les cardinalités du côté de l'entité 'ouvrage' :

- Un ouvrage peut-il ne pas avoir d'auteur ? Si c'est le cas, la valeur minimale sera de '0' ; sinon, elle sera de '1'. On choisit '1' : on suppose qu'il n'y a pas de livre anonyme.

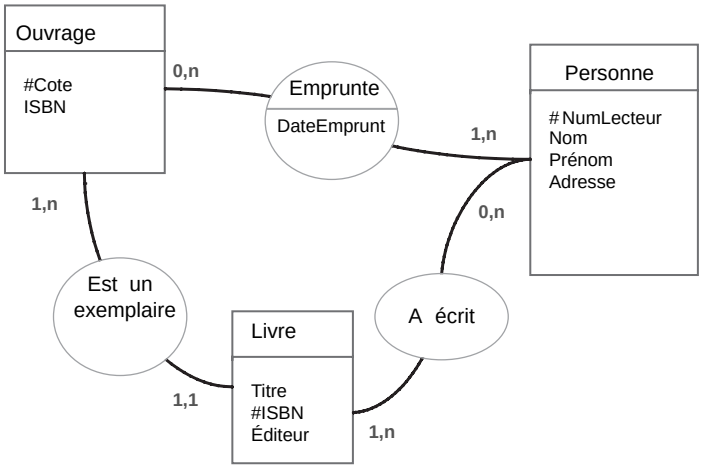
- Un ouvrage peut-il avoir plusieurs auteurs ? Si c'est le cas, la valeur maximale sera de 'n' ; sinon, elle sera de'1' : un livre peut avoir plusieurs auteurs ; c'était d'ailleurs la motivation pour créer l'entité 'auteur'. On choisit donc 'n'.

Pour déterminer les cardinalités du côté de l'entité 'personne' :

- Une personne peut-elle ne pas avoir écrit de livre ? Si c'est le cas, la valeur minimale sera de '0' ; sinon, elle sera de '1'. On choisit '0' : une personne peut ne pas être un auteur.
- Une personne peut-elle être l'auteur de plusieurs livres ? Si c'est le cas, la cardinalité maximale sera de 'n' ; sinon, elle sera de'1'. On choisit 'n' : un auteur peut écrire plusieurs livres.

Figure 2.22

Entités 'livre', 'ouvrage', 'personne' liées par les associations 'emprunte', 'a_écrit' et 'est_un_exemplaire' (avec cardinalités).



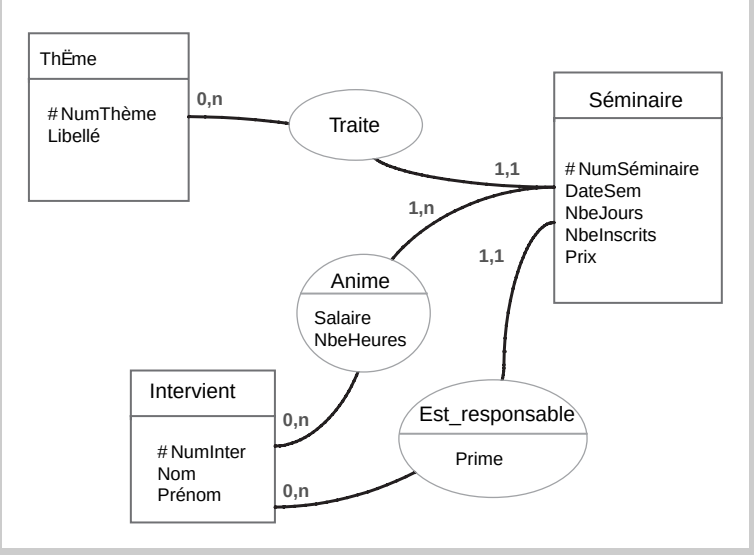
EXERCICE 4 DESCRIPTION DU MONDE RÉEL À PARTIR DES CARDINALITÉS

Énoncé

On considère le schéma entité-association, muni de ses cardinalités, qui décrit une partie de l'organisation de séminaires (voir figure 2.23).

Figure 2.23

Entités 'séminaire', 'thème', 'intervenant' liées par les associations 'traite', 'est_responsable' et 'intervient' (avec cardinalités).



Énoncé Quelle description pouvez-vous donner du lien entre les différentes entités à partir des cardinalités ?

Solution Un séminaire traite d'un thème et d'un seul (cardinalité 1-1). Un thème peut être traité par aucun séminaire ou par plusieurs (cardinalité 0-n).

Un séminaire a un intervenant au minimum (cardinalité 1-n). Un intervenant peut intervenir dans plusieurs séminaires ou aucun (cardinalité 0-n).

Un séminaire a toujours un responsable et un seul (cardinalité 1-1). Un intervenant peut n'être responsable d'aucun séminaire ou l'être de plusieurs (cardinalité 0-n).

EXERCICE 5 ASSOCIATION INUTILE

Énoncé On décrit une (partie de la) réalité biologique d'un système parasite-hôte de la manière suivante :

- Un parasite utilise un et un seul type d'hôte.
- Un hôte a un et un seul parasite.

Décrivez les entités et les associations que vous identifiez à partir de cette description et déduisez-en les cardinalités associées. Que proposez-vous pour améliorer le schéma ?

Solution On identifie aisément les entités parasite et hôte comme les sujets ou les compléments des phrases de type « sujet-verbe-complément » qui décrivent le système. Les cardinalités seront de type « 1-1 » de chaque côté : chaque hôte est associé de manière unique à un parasite et inversement. Les associations de cardinalités '1-1' de chaque côté sont généralement inutiles : il est préférable de les remplacer par une entité unique même si l'on perd en lisibilité au niveau du schéma. On fusionne les entités 'parasite' et 'hôte' en une seule entité 'écosystème' (voir figure 2.24).

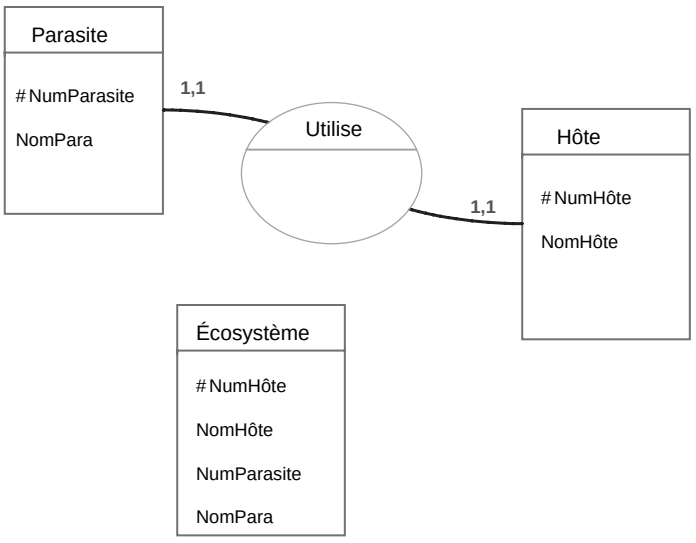


Figure 2.24

Entités parasite-hôte liées par l'association 'utilise' et fusion en une entité unique.

EXERCICE 6 ASSOCIATION RÉFLEXIVE

Énoncé

On veut représenter les liens de nourriture entre des humains, des animaux et des végétaux. L'idée, à partir des schémas d'alimentation modélisés, est de pouvoir déduire des chaînes alimentaires de ce type : « un homme mange un lapin qui mange des carottes ».

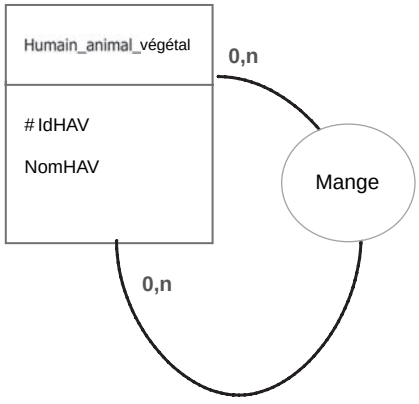
Solution

On pourrait différencier les humains/animaux des végétaux et créer deux entités différentes. Mais comme les humains mangent des animaux et des végétaux et que les animaux mangent également d'autres animaux et des végétaux, il n'est pas pertinent de les séparer pour représenter ce type d'information. On utilise une seule entité 'humain_animal_végétal' qui est liée à elle-même par l'association 'mange'. Les cardinalités sont de type '0-n' (voir figure 2.25) :

- Un humain_animal_végétal peut manger ou ne pas en manger un autre (un végétal ne mange pas ses congénères).
- Un humain_animal_végétal peut être mangé ou ne pas être mangé par un autre.

Figure 2.25

Entité 'humain_animal_végétal' liée par l'association 'mange' (avec cardinalités).



EXERCICE 7 ASSOCIATION TERNAIRE

Énoncé

À partir de la base de données exemple de vente de voitures, on souhaite ajouter les informations concernant le vendeur qui a réalisé la vente. Proposez une (ou plusieurs) modification(s) du modèle entité-association élaboré précédemment. Ajoutez les nouvelles cardinalités introduites par cette modification.

Solution

On doit définir une nouvelle entité 'vendeur'. En effet, les informations du vendeur sont indépendantes de celles des voitures ou des clients. On peut envisager le cas où l'on ajoute les informations du vendeur comme attributs de l'association 'vente', mais un vendeur est susceptible de réaliser plus d'une vente. On répéterait alors ces informations, identiques pour le même vendeur, avec les risques d'incohérence classiques.

L'entité 'vendeur' est liée à l'entité 'voiture' mais également à l'entité 'client'. Ces liens sont créés par l'opération de vente :

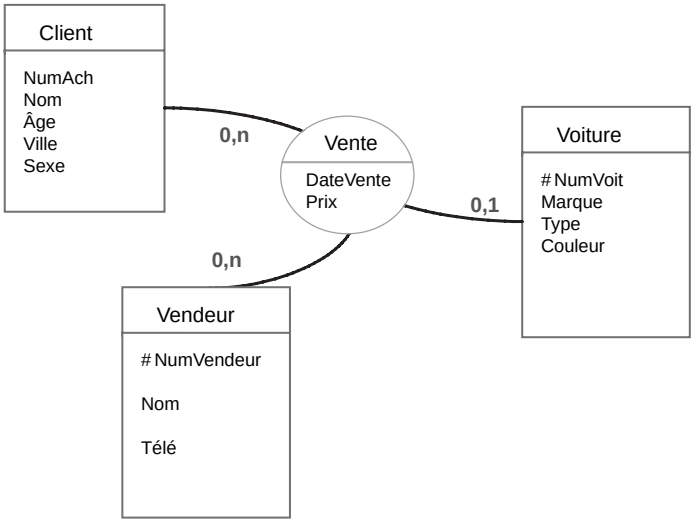
- Un vendeur vend une voiture.
- Un vendeur vend à un client.

Cette nouvelle entité sera liée aux deux autres par l'association 'vente'. On se trouve dans le cas où l'association sera donc non plus de type binaire, mais ternaire. Les cardinalités associées aux entités 'voiture' et 'client' ne changent pas ; elles sont de type '0-n' (« une personne peut acheter plusieurs voitures ou aucune » ; « une voiture peut être vendue une seule fois ou jamais »).

Pour une association de type ternaire, on emploie plus ou moins implicitement des cardinalités de type '0-n'. Cela correspond du côté du vendeur à une phrase un peu ambiguë de la forme : « un vendeur peut vendre aucune ou plusieurs voitures à aucun ou plusieurs clients ». Comme on utilise la notation européenne, on n'effectue pas de changements du côté des entités 'client' et 'voiture' (voir figure 2.26).

Figure 2.26

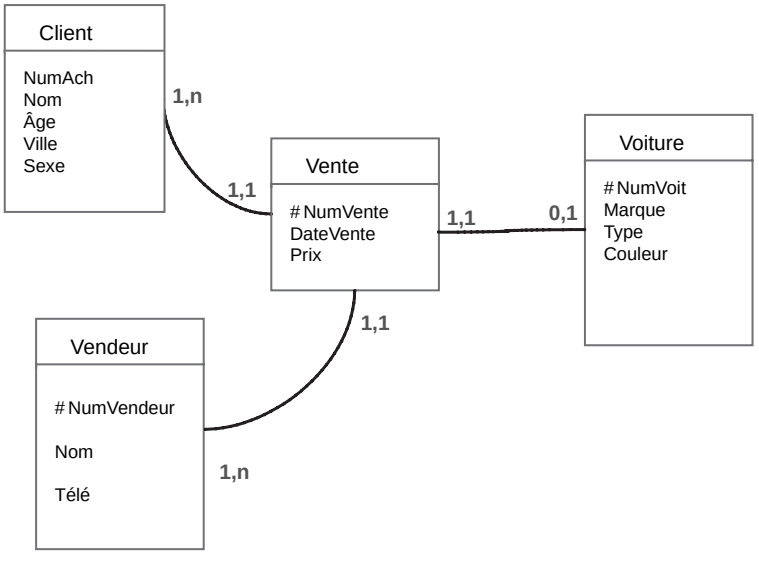
Entités 'voiture', 'client' et 'vendeur' liées par les associations ternaire 'vente' (avec cardinalités).



Les associations ternaires sont parfois délicates à utiliser et difficiles à représenter en UML sans contorsions. De plus, les cardinalités perdent de leur pertinence dans ce contexte. Il est souvent préférable de remplacer cette association par une entité que l'on relie aux autres par des associations binaires.

Figure 2.27

Entités 'voiture', 'client', 'vendeur' et 'vente' liées par des associations binaire après transformation de l'association ternaire 'vente' en entité (avec cardinalités).



EXERCICE 8 DE L'ÉNONCÉ AU MODÈLE ENTITÉ-ASSOCIATION

Énoncé

Vous désirez gérer un club de prêt de DVD. Les clients (adhérents du club) versent une somme sur leur compte lors de leur adhésion. Ils peuvent réserver le film avant de le louer et peuvent le garder une semaine au maximum. Le prix de location du film est forfaitaire par jour emprunté. Il leur est possible de se faire livrer le film chez eux : cette opération est facturée forfaitairement en plus.

Le film est décrit par son titre, le genre du film, le réalisateur et les trois acteurs principaux. On précise également le nombre de DVD (il peut y en avoir plusieurs dans la pochette : « making of », autres versions, etc.) et leur prix d'achat qui permettra de débiter le compte du client en cas de non-retour du film.

Vous voulez pouvoir annuler et mettre à jour les réservations et gérer les comptes des adhérents (par exemple ne plus prêter au-delà d'un certain seuil…). Vous vous servirez également de cette base de données pour effectuer des bilans (tels que le chiffre d'affaires en fin de mois…), des relances (les films non rendus à temps…) et des statistiques (film le plus emprunté, meilleur client…).

Proposez un modèle entité-association pour cette activité.

Solution

On peut extraire de l'énoncé des phrases clés qui permettent de caractériser les entités et leurs associations :

- Un client réserve un film.
- Un client loue un film.
- Une personne joue dans un film.
- Une personne réalise un film.

Trois entités apparaissent : 'client', 'film' et 'personnel'. Les entités 'client' et ' film' sont reliées par deux associations : 'réservation' et 'location'. Les entités 'film' et 'personnel' sont reliées par deux entités 'joue' et 'réalise' (voir figure 2.28).

Le fait d'utiliser la même entité pour un réalisateur et pour un acteur (regroupés dans 'personnel') provient du fait qu'un acteur peut en même temps être réalisateur. Si l'on séparait les entités, il y aurait alors une duplication de l'information concernant cette catégorie 'acteurs/réalisateurs'. Le reste de l'énoncé permet de décrire les attributs des entités et des associations.

Entité 'client'

On a peu d'informations sur cette entité, si ce n'est que le client possède un compte qui fera partie de ses attributs. Les autres attributs ont été choisis classiquement (nom, prénom, adresse, tél., etc.). Il est nécessaire d'utiliser un attribut pour l'identifiant : ici, 'NumClient'.

Entité 'film'

L'entité est mieux décrite par son titre, son genre, son prix et le nombre de DVD dans la pochette. Il est également nécessaire d'utiliser un attribut pour l'identifiant : ici, 'NumFilm'.

Entité 'personnel'

L'entité ne comprend que les informations nom et prénom. Il est également nécessaire de créer un attribut pour l'identifiant : ici, 'NumPers'.

Association 'réservation'

L'association a pour attribut la date de réservation et le nombre de jours de réservation. On a ajouté un champ identifiant de la réservation 'NumRés'. Il n'est pas absolument nécessaire d'un point de vue du modèle, une association n'a pas besoin d'un attribut identifiant, mais il facilitera la gestion.

Association 'location'

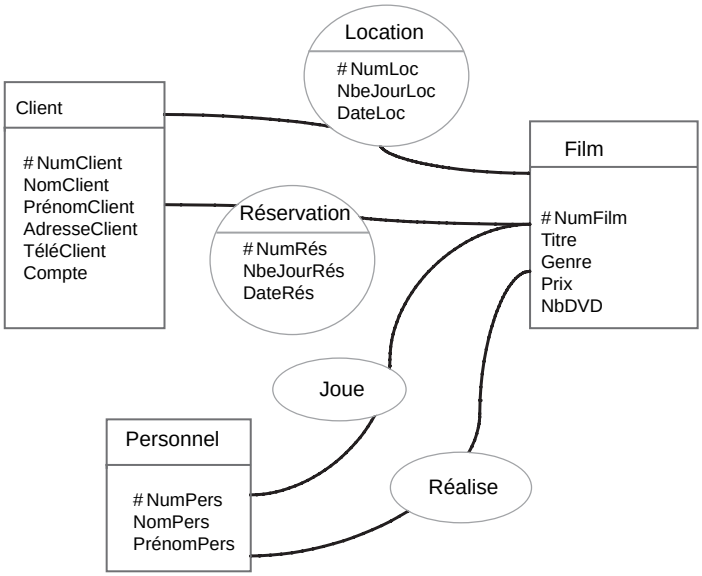
L'association a pour attribut la date et le nombre de jours de location. On a ajouté un champ identifiant de la location, 'NumLoc', pour les mêmes raisons que précédemment. L'attribut 'livraison' indique si la livraison a été effectuée, afin le cas échéant de la facturer.

Associations 'joue' et 'réalise'

Les deux associations n'ont pas d'attributs.

Figure 2.28

Entités 'clients', 'film, 'personnel' liées par les associations 'réservation', 'location', 'joue' et 'réalise' (sans cardinalités).



EXERCICE 9 REPRÉSENTATION AVEC UML

Énoncé

À partir du modèle entité-association modélisant le club de location de DVD précédent, faites sa représentation en utilisant UML. Proposez des cardinalités pour les associations en les justifiant.

Solution

Les entités sont représentées par des classes UML. Les associations sont représentées par des associations UML. Les associations 'location' et 'réservation' qui possèdent des attributs seront munies d'une classe UML sans nom qui permet d'utiliser ces attributs. Les cardinalités sont déduites des phrases suivantes :

Location :

- Un client loue aucun ou plusieurs films (0..n).
- Un film est loué aucune ou plusieurs fois (0..n).

Réservation :

- Un client réserve aucun ou plusieurs films (0..n).

- Un film est réservé aucune ou plusieurs fois (0..n).

Joue :

- Une personne joue dans aucun ou plusieurs films (0..n).
- Un film a au moins un acteur (1..n).

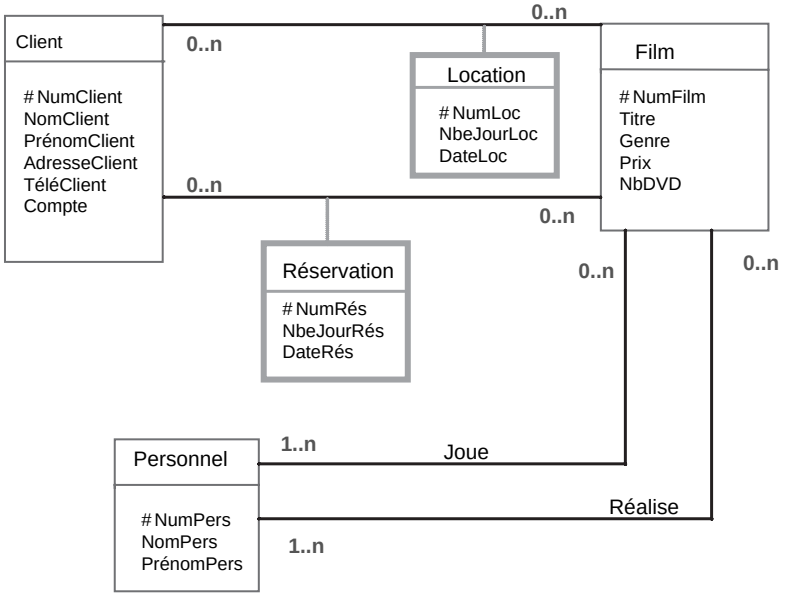
Réalise :

- Une personne réalise aucun ou plusieurs films (0..n).
- Un film a un et un seul réalisateur (1..1).

Enfin, la notation des cardinalités est inversée par rapport au modèle entité-association (voir figure 2.29).

Figure 2.29

Représentation UML des entités 'clients', 'film', 'personnel' liées par les associations 'reservation', 'location', 'joue' et 'réalise' (avec cardinalités).



EXERCICE 10 AUTRE EXEMPLE – LE CAMPING L'ULIASTRU

Énoncé

Le camping l'Uliastru (l'olivier sauvage), situé dans l'extrême sud de la France, propose à ses clients différents types de locations : des « bungalows toile » pour 350 €/semaine, des caravanes pour 440 €/semaine, des tentes pour 45 €/jour. Les différentes formules offrent un équipement complet et appartiennent au camping. Il est également possible de louer un « emplacement tourisme » à la journée pour 28 €/personne. L'ensemble de ces locations s'adresse à un maximum de quatre personnes. Proposez un modèle entité-association modélisant cette activité de gestion en fonction des éléments de l'énoncé.

Solution

Voici les phrases clés déduites de l'énoncé qui permettent de caractériser les entités et les cardinalités des associations du modèle entité-association :

- Les clients louent des séjours, ils peuvent en louer plusieurs (1..n).
- Les séjours peuvent être loués par un ou plusieurs clients (du moment que les périodes de location ne se chevauchent pas) (0,n).

On en déduit les entités 'Séjour' et 'Client', ainsi que l'association 'Louer'. La première difficulté que l'on rencontre est liée à la notion de temps. En effet, dans quelle entité doit être stockée la notion de durée de la location nécessaire à la gestion éventuelle des disponibilités ? D'après l'énoncé, la durée de la location est fonction du type de location (bungalows toile : 7 jours, caravanes : 7 jours, tentes : 1 jour, emplacements tourisme : 1 jour). Le client ne peut donc pas choisir la durée minimum ; cette dernière est déterminée en fonction du type de location. Par conséquent, il convient de stocker l'attribut 'duree_mini' modélisant la durée de location minimum dans l'entité 'location'. L'unité commune à l'ensemble des locations pour la facturation est la journée ; les durées s'expriment alors en nombre de jours.

La seconde difficulté est liée à la notion de personne. Comment intervient-elle dans le schéma conceptuel ? Tout d'abord, remarquons que, d'après la première phrase clé, un client peut représenter plusieurs personnes car il peut louer plusieurs locations. Par conséquent, le moyen d'intégrer efficacement la limite de quatre personnes et les quantités nécessaires à la facturation des différents types de location consiste en l'ajout de l'attribut 'quantité' dans l'association 'Louer'. Le client du modèle représente la personne physique que l'on facture.

Finalement, voici le MCD de cette activité de gestion :

- Séjour(<u>Type</u>, Durée_mini, Prix_ttc)
- Client(<u>ID</u>, Nom, Prénom, Adresse, Ville, CP, Téléphone)
- Louer(quantité, date_début)

La date de début de location dépend à la fois du client et de la location ; c'est la raison pour laquelle elle apparaît dans l'association 'Louer'. La date de fin d'une location est calculée à partir de la durée minimale et de la quantité louée en fonction de la date de début de location ; c'est la raison pour laquelle il est inutile de la stocker.

Chapitre 3

Approche relationnelle

L'approche relationnelle présentée par E. F. Codd possède un fondement mathématique rigoureux qui lui assure sa robustesse : le modèle relationnel est de loin le plus répandu dans le monde des bases de données.

Ce chapitre présente le concept de relation fondamental du modèle relationnel, ainsi que les opérations qui lui sont associées. Puis on aborde les méthodes qui permettent de passer du modèle conceptuel vu au chapitre précédent (entité-association ou UML) à un ensemble de relations. La qualité des relations ainsi produites est contrôlée par leur conformité aux trois premières formes normales. Ce processus peut conduire à une réorganisation des relations sans perte d'information. La manière de résoudre les incohérences à l'aide de la forme normale de Boyce-Codd est également présentée. Enfin, le lien entre les bases de données et la logique du premier ordre est rapidement exposé, afin de décrire la méthode d'interrogation de base de données par QBE (*Query By Example*) qui en découle.

1 Concepts

Cette section expose la notion de relation et la terminologie qui lui est associée. On présente ensuite les différents types de contraintes associées au contenu d'une relation, ainsi que la notion de clé d'une relation.

1.1 Modèle relationnel

Le modèle relationnel tire son nom de la notion de **relation** mathématique entre des éléments. Chacun de ces éléments peut prendre des valeurs dans un ensemble défini.

On suppose que l'on considère les appareils electroménagers d'une cuisine. Ils peuvent être contenus dans l'ensemble des valeurs suivantes : *réfrigérateur*, *cuisinière*, *hotte*, *robot*, *lave-vaisselle*. On considère par ailleurs un ensemble de couleurs qui peuvent être contenues dans l'ensemble des valeurs suivantes : *rouge*, *bleu*, *vert*, *jaune*, *blanc*, *noir*, *rose*, *jaune*. Les combinaisons possibles entre les appareils et les couleurs sont au nombre de 40, puisqu'il y a 5 appareils que l'on peut associer à 8 couleurs. Parmi toutes ces combinaisons possibles, on effectue une sélection qui représente par exemple la description d'une (horrible) cuisine dans le monde réel. Ces couples de valeurs choisis représentent les **faits** de la vie réelle.

```
(réfrigérateur, rouge)
(robot,mauve)
(cuisinière,jaune)
(lave-vaisselle,rouge)
```

Cet ensemble de couples de valeurs liées entre elles, que l'on nomme **tuples** dans le modèle relationnel, représente la **relation** entre les éléments 'appareil' et 'couleur'. Un tuple est aussi désigné par les termes « **nuplets** » ou « **enregistrements** ». On désigne également les éléments constitutifs de ces couples par les termes « **attributs** » ou « **champs** ».

On peut écrire formellement la relation de la manière suivante : **ma_cuisine(appareil, couleur)**. Cette écriture représente le **schéma relationnel** de la relation 'ma_cuisine'. Les valeurs énoncées précédemment pour les champs représentent leurs **domaines**, c'est-à-dire les ensembles de toutes les valeurs possibles pour un champ.

Une relation est totalement décrite par :

- le schéma relationnel ;
- les domaines des différents champs ;
- les tuples qui la constituent.

Le nombre de champs de la relation s'appelle son **degré de la relation**. Ici, la relation 'ma_cuisine' est de degré 2. Le nombre de tuples se nomme la **cardinalité** de la relation. La relation 'ma_cuisine' est de cardinalité 4. Attention, il ne s'agit pas de la même cardinalité que pour le modèle entité-association vu précédemment.

On représente une relation par une **table**, correspondant à la notion de tableau. Les tuples correspondent aux lignes et les colonnes aux champs de la relation. Voici sous forme de table une représentation de l'exemple précédent (voir figure 3.1).

Dans le modèle relationnel, la relation est l'élément fondamental. Toutes les opérations sur une ou plusieurs relations retourneront une relation. Un ensemble de relations reliées entre elles par des liens sémantiques constitue une **base de données**.

Figure 3.1

Table 'ma_cuisine'.

Appareil	Couleur
Réfrigérateur	Rouge
Robot	Mauve
Cuisinière	Jaune
Lave-vaisselle	Rouge

1.2 Notion de clé d'une relation et dépendance fonctionnelle

Lorsque que l'on utilise une base de données, il est nécessaire d'accéder à un enregistrement par le contenu d'un ou de plusieurs champs. On nomme **clé** d'une relation un champ, ou un ensemble de champs, d'une relation qui permet d'identifier précisément un enregistrement. Une relation peut comprendre plusieurs clés possibles ; ce sont les **clés candidates**. La clé choisie doit être minimale, c'est-à-dire qu'elle doit contenir le minimum de champs. Cette clé minimale ainsi définie est appelée la **clé primaire** de la relation. Une clé doit toujours contenir une valeur et celle-ci doit être unique pour chacun des enregistrements de la relation.

La clé ne peut être déduite simplement à partir du contenu de la relation ; on ne peut préjuger du contenu futur des enregistrements. Si l'on prend la relation 'ma_cuisine' vue précédemment, le champ 'Appareil' semble être une clé puisqu'il contient une valeur unique pour chacun des enregistrements. Cependant, il est tout à fait possible que la cuisine considérée comprenne un autre réfrigérateur de couleur bleue, auquel cas la valeur ne serait plus unique et ne permettrait pas de retrouver l'enregistrement. Dans ce cas de figure, la combinaison 'Appareil' 'Couleur' pourrait sembler être une clé, mais on ne peut en être certain compte tenu de l'évolution des données.

Pour désigner une clé primaire, il faut donc également prendre en compte le « sens » des données dans la vie réelle. Les relations qui existent entre les différents champs d'une relation vont être importantes : on exprime ces relations à l'aide de **dépendances fonctionnelles**. Une dépendance fonctionnelle existe entre deux ensembles de champs si les valeurs contenues dans l'un des ensembles de champs permettent de déterminer les valeurs contenues dans l'autre ensemble.

Cette propriété se traduit en termes mathématiques de la manière suivante :

```
Soit Cx un ensemble de champs (x1,x2,x3) et Cy un ensemble de champs (y1,y2,y3)
d'une relation R(d1,d2,x1,x2,x3,u1,u2,u3,y1,y2,y3).
Supposons que l'on considère des valeurs (x1_i, x2_i, x3_i) et (x1_j, x2_j, x3_j)
telles que l'on ait R(d1_i,d2_i,x1_i,x2_i,x3_i,u1_i,u2_i,u3_i,y1_i,y2_i,y3_i) et
R(d1_j,d2_j,x1_j,x2_j,x3_j,u1_j,u2_j,u3_j,y1_j,y2_j,y3_j).
On dit que Cy dépend fonctionnellement de Cx lorsque pour tout i et j si (x1_i, x2_i,
x3_i) est égal à (x1_j, x2_j, x3_j) alors (y1_i,y2_i,y3_i) est égal à
(y1_j,y2_j,y3_j).
```

Les dépendances fonctionnelles expriment la relation de hiérarchie qui existe entre les champs.

On considère l'exemple de table suivant (voir figure 3.2), qui correspond à la relation Lecteur(Numero_carte, Nom, Age, Ville, Etablissement). Cet exemple modélise les lecteurs d'une bibliothèque.

Figure 3.2

Relation 'Lecteur'.

Numero _carte	Nom	Age	Ville	Etablissement
1	Henri	10	Paris	Université Sorbonne
2	Stanislas	34	Paris	Université Jussieu
3	Henriette	44	Lyon	CHU Bron
4	Dominique	19	Nancy	Université Poincaré
5	Isabelle	56	Nancy	INPL
6	Olivier	51	Marseille	Université Saint Charles
7	Henri	98	Paris	Université Sorbonne
8	Jerome	23	Nancy	INPL
9	Laurence	34	Bordeaux	Université Victor Segalen
10	Christian	41	Paris	Ecole Normale Supérieure
11	Antoine	16	Marseille	Université Saint Charles
12	Laurence	34	Paris	Université Jussieu

Si l'on examine les données, on remarque qu'il ne peut y avoir de dépendances fonctionnelles entre les couples de champs (Ville, Etablissement) et les champs (Nom, Age). Il existe un enregistrement ('Laurence', '34') pour lequel les valeurs des champs (Nom, Age) correspondent à deux valeurs différentes de (Ville, Etablissement).

En revanche, on sait que, dans la réalité, un établissement est situé dans une ville et une seule (on le suppose pour cet exemple). Cela signifie qu'il existe une relation de dépendance entre les champs 'Etablissement' et 'Ville'. Le contenu des champs 'Ville' et 'Etablissement' des enregistrements de notre relation se conforment à cette relation de dépendance. A une valeur donnée de 'Etablissement' correspond bien une valeur unique de 'Ville'.

La valeur du champ 'Numero_carte' est unique pour chacune des personnes. On constate que ses valeurs sont identifiantes pour tous les autres champs de la relation. Chaque champ dépend fonctionnellement du champ 'Numero_carte'. Ses valeurs sont uniques et jamais vides : c'est une clé candidate. Dans cet exemple, c'est la seule clé possible car les autres champs n'ont jamais de valeur unique. Le champ' 'Numero_carte' est choisi comme clé primaire de la relation.

Remarque

Tous les champs qui ne font pas partie d'une clé candidate d'une relation possèdent des dépendances fonctionnelles avec cette clé.

Les dépendances entre les champs représentent les liens entre les différents éléments du monde réel. On rappelle qu'elles ne peuvent être déduites du contenu de la relation, même si cela peut constituer un point de départ. Elles sont donc déterminées durant la phase précédente d'analyse du monde réel. Il est possible en revanche de vérifier si les enregistrements présents dans la relation se conforment à ces dépendances. La détermination des dépendances fonctionnelles entre les champs est fondamentale pour l'étape de normalisation, traitée à la section « Normalisation ». Elles sont également à la base de la méthode dite de la « relation universelle », qui sera abordée dans la section « Normalisation ».

1.3 Cohérence des données et contraintes d'intégrité

Afin de garantir la cohérence des données pour l'ensemble des relations constitutives de la base de données, on applique des restrictions sur le contenu des données que l'on nomme **contraintes d'intégrité**. La vérification de la cohérence se situe à plusieurs niveaux :

- adapter le contenu des champs par rapport au sens des données dans le monde réel ;
- préserver la cohérence du contenu de l'ensemble des relations qui sont liées ;
- éliminer les problèmes d'incohérence dus à la redondance : en effet la duplication des données rend délicates la maintenance et l'évolution de la relation.

Adapter le contenu des données

La cohérence par rapport au sens des données est liée à la notion de domaine de valeurs du champ déjà abordée : on parle de **contrainte de domaine** ou **cohérence sémantique**. Par exemple, l'âge d'une personne ne peut être négatif ou excéder 120. On a précédemment défini une contrainte d'intégrité sur l'ensemble des champs qui constituent une clé primaire : une clé ne peut contenir de valeur nulles et ses valeurs doivent être uniques. On procède à la définition des contraintes lors de l'étape d'analyse du monde réel. Leur mise en œuvre effective interviendra au moment de la création de la relation et sera réalisée par le SGBD.

Préserver la cohérence des données

On rappelle qu'une base de données est constituée par un ensemble de relations reliées entre elles. Les contenus des champs capables de lier ces relations doivent être cohérents entre eux pour pouvoir effectuer l'opération de jointure. Si l'on considère l'exemple de la base de données exemple 'casse', on ne doit pas permettre la saisie d'une valeur identifiant une voiture dans la relation 'vente' qui n'existe pas dans la relation 'voiture'. On fait dans ce cas référence au contenu d'une colonne d'une autre relation, le champ 'NumVoit' de la relation voiture, pour contrôler le contenu du champ 'NumVoit' de la relation 'vente'. Le champ 'NumVoit' est alors une **clé étrangère** qui permet de réaliser la notion d'**intégrité référentielle**. De même que précédemment, on détermine ces références lors de la phase d'analyse du monde réel et le SGBD permet de les réaliser. On préfère appliquer en général ces contraintes non pas lors de la création des relations, mais plutôt lorsque les données ont déjà été insérées, en particulier dans les relations de références. Cette précaution évite les problèmes de références impossibles à résoudre entre les relations, ce qui provoque parfois l'incapacité à insérer des données.

Éliminer la redondance des données

Les incohérences provoquées par la **redondance** d'information représentent le principal souci du concepteur d'une base de données. En effet, lorsque les données sont dupliquées, aucun mécanisme ne peut garantir que le changement de la valeur d'une donnée est répercuté correctement sur les autres données. Dans l'exemple ci-dessus (voir figure 3.2) d'une relation des lecteurs de bibliothèque, on remarque la redondance d'information qui existe entre les champs 'Ville' et 'Etablissement'. Si le nom de l'établissement 'INPL' change, il faut mettre à jour toutes les lignes qui contiennent son nom. La redondance est mise en évidence par la dépendance fonctionnelle qui existe entre ces deux champs. Dans cet exemple, détecter la redondance est évident. Cependant, il s'agit généralement d'incohérences délicates à déceler lorsque le nombre de relations est élevé ou encore si le sujet modélisé par la base de données n'est pas familier. Les incohérences de ce type seront résolues par la réorganisation des relations lors de la phase de normalisation.

2 Opérations du modèle relationnel

La manipulation des données dans le modèle relationnel se fait à l'aide d'opérations formelles reposant sur des concepts mathématiques issus de la théorie des ensembles : c'est **l'algèbre relationnelle**. Les opérations de l'algèbre relationnelle portent sur une ou plusieurs relations (ou tables). Le résultat retourné par ces opérations est toujours une relation (ou table).

Cette section présente une liste non exhaustive des différentes opérations de l'algèbre relationnelle. Ces dernières peuvent être regroupées en plusieurs catégories :

- les opérations classiques issues de la théorie des ensembles (union, intersection, différence) ;
- les opérations plus spécifiques du monde relationnel (projection, sélection, jointure) qui constituent les opérations fondamentales de l'algèbre relationnelle ;
- les opérations de types calcul et agrégats qui ne constituent pas réellement des opérations fondamentales (ni ensemblistes, ni relationnelles) mais qui sont le complément indispensable des précédentes.

2.1 Opérations ensemblistes

L'algèbre relationnelle emprunte un certain nombre de concepts à la théorie des ensembles. À l'exception du produit cartésien, ces opérations ont la particularité de ne pouvoir s'utiliser que pour des relations qui ont exactement la même structure ou des structures compatibles. Elles sont toutes de type binaire, car elles s'appliquent à deux relations.

Union

L'opération d'union consiste en la mise en commun des enregistrements de chaque relation. Les enregistrements identiques ne sont intégrés qu'une seule fois. Dans l'exemple ci-après (voir figures 3.3 et 3.4), le tuple dont la valeur du champ 'Numero_carte' vaut 3 ne sera pas dupliqué. L'union est représentée par le caractère « ∪ ». Cette opération sert typiquement à la « consolidation » de données de même type provenant de différentes sources.

Figure 3.3

Relations 'Lecteur_1' et 'Lecteur_2'.

Lecteur_1(Numero_carte, Nom, Age, Ville, Etablissement)

Numero_carte	Nom	Age	Ville	Etablissement
1	Henri	10	Paris	Université Sorbonne
2	Stanislas	34	Paris	Université Jussieu
3	Henriette	44	Lyon	CHU Bron

Lecteur_2(Numero_carte, Nom, Age, Ville, Etablissement)

Numero_carte	Nom	Age	Ville	Etablissement
3	Henriette	44	Lyon	CHU Bron
4	Dominique	19	Nancy	Université Poincaré
5	Isabelle	56	Nancy	INPL

Figure 3.4 **Lecteur_1 ∪ Lecteur_2**

Union des relations 'Lecteur_1' et 'Lecteur_2'.

Numero_carte	Nom	Age	Ville	Etablissement
1	Henri	10	Paris	Université Sorbonne
2	Stanislas	34	Paris	Université Jussieu
3	Henriette	44	Lyon	CHU Bron
4	Dominique	19	Nancy	Université Poincaré
5	Isabelle	56	Nancy	INPL

Différence

L'opération différence consiste à désigner les enregistrements qui appartiennent à une relation sans appartenir à l'autre. La différence est représentée par le caractère « – » (voir figure 3.5). Attention, cette opération n'est pas symétrique ; le résultat de l'opération 'Lecteur_1 – Lecteur_2' est en général différent de 'Lecteur_2 – Lecteur_1'. Elle permet par exemple d'éliminer des enregistrements d'une relation par rapport à une liste.

Figure 3.5 **Lecteur_1 – Lecteur_2**

Différence des relations 'Lecteur_1' et 'Lecteur_2'.

Numero_carte	Nom	Age	Ville	Etablissement
1	Henri	10	Paris	Université Sorbonne
2	Stanislas	34	Paris	Université Jussieu

Intersection

L'opération intersection peut se déduire de la précédente ; elle désigne les enregistrements qui sont communs aux deux relations. L'intersection est représentée par le caractère « ∩ » (voir figure 3.6). Elle permet de trouver les éléments communs à deux relations.

Figure 3.6 **Lecteur_1 ∩ Lecteur_2**

Intersection des relations 'Lecteur_1' et 'Lecteur_2'.

Numero_carte	Nom	Age	Ville	Etablissement
3	Henriette	44	Lyon	CHU Bron

Produit cartésien

Le produit cartésien permet la combinaison des enregistrements de deux relations sans tenir aucun compte du contenu des données. Les relations n'ont donc pas besoin d'avoir la même structure. Le caractère représentant le produit cartésien est « X ».

Figure 3.7 **Relations 'ma_cuisine' et 'musicien'.**

ma_cuisine(Appareil, Couleur)

Appareil	Couleur
Réfrigérateur	rouge
Robot	mauve
Cuisinière	jaune

musicien(Nom, Instrument)

Nom	Instrument
Jaco Pastorius	Basse électrique
Bill Evans	Piano

L'exemple a été choisi de manière à montrer que les relations ne nécessitent pas de rapports entre elles pour faire un produit cartésien. Combiner des appareils de cuisine et des musiciens n'a aucun sens dans la réalité (voir figures 3.7 et 3.8).

Figure 3.8 **Produit cartésien des relations 'ma_cuisine' et 'musicien'.**

Le_resultat(Appareil, Couleur, Nom, Instrument) = ma_cuisine(Appareil, Couleur) X musicien(Nom, Instrument)

Appareil	Couleur	Nom	Instrument
réfrigérateur	rouge	Jaco Pastorius	Basse électrique
réfrigérateur	rouge	Bill Evans	Piano
robot	mauve	Jaco Pastorius	Basse électrique
robot	mauve	Bill Evans	Piano
cuisinière	jaune	Jaco Pastorius	Basse électrique
cuisinière	jaune	Bill Evans	Piano

2.2 Opérations relationnelles

Projection

La projection consiste à extraire certains champs de la relation, ce qui donne à cette dernière un degré inférieur à la relation de départ. Voici la relation obtenue à partir de la relation 'Lecteur' en projetant les champs 'Nom' et 'Ville' (voir figure 3.9).

Figure 3.9 **Projection des champs 'Nom' et 'Ville' de la relation 'Lecteur'.**

Lecteur_proj(Nom, Ville)

Nom	Ville
Henri	Paris
Stanislas	Paris

Nom	Ville
Henriette	Lyon
Dominique	Nancy
Isabelle	Nancy
Olivier	Marseille
Henri	Paris
Jerome	Nancy
Laurence	Bordeaux
Christian	Paris
Antoine	Marseille
Laurence	Paris

Intuitivement, dans une représentation de type table, on conserve uniquement les colonnes sur lesquelles la projection est faite.

Sélection ou restriction

La sélection consiste à extraire les enregistrements de la relation. On utilise des critères pour caractériser les enregistrements sélectionnés. Voici la relation obtenue à partir de la relation 'Lecteur 'en sélectionnant les enregistrements dont le contenu du champ 'Ville' est 'Marseille' (voir figure 3.10). La structure de la relation résultat est la même que celle de la relation de départ.

Figure 3.10 Sélection sur la relation 'Lecteur'.

Lecteur_sel(Numero_carte, Nom, Age, Ville, Etablissement)

Numero_carte	Nom	Age	Ville	Etablissement
6	Olivier	51	Marseille	Université Saint Charles
11	Antoine	16	Marseille	Université Saint Charles

Intuitivement, dans une représentation de type table, on conserve uniquement les lignes répondant au critère.

Jointure

La jointure est l'opération fondamentale de l'algèbre relationnelle qui permettra d'exprimer le sens du lien entre les relations dans le monde réel. La liaison entre les relations s'effectue par le contenu commun d'un champ. L'opération de jointure peut être vue comme une sélection des enregistrements obtenus par le produit cartésien des relations, dont les contenus du champ sur lequel on effectue la jointure sont égaux. On l'appelle dans ce cas une **équijointure**. Les champs dont on compare les contenus sont nommés **champs de jointure**.

On considère les deux relations Lecteur_bis(Numéro_carte, Nom, Num_Etablissement) et Etablissement(Num_Etablissement, Ville, Nom_Etablissement) dont le contenu suit (voir figure 3.11).

Figure 3.11 **Lecteur_bis(Numero_carte, Nom, Num_Etablissement)**

Relations 'Lecteur_bis' et 'Etablissement'.

Numero_carte	Nom	Num_Etablissement
1	Henri	1
2	Stanislas	2
3	Henriette	1

Etablissement(Num_Etablissement, Ville, Nom_Etablissement)

Num_Etablissement	Ville	Nom_Etablissement
1	Paris	Université Jussieu
2	Lyon	CHU Bron
3	Nancy	Université Poincaré
4	Paris	Université Sorbonne

Même si l'on ne dispose pas du modèle conceptuel associé, on constate que l'on peut relier ces deux relations par le champ 'Num_Etablissement'. Les informations concernant l'établissement de la relation Lecteur_bis sont stockées dans la relation Etablissement dont la clé est le champ Num_etablissement. Pour obtenir la liste des lecteurs ainsi que les informations concernant leur établissement, on effectue une jointure entre ces relations sur le champ 'Num_Etablissement' (voir figure 3.12).

Figure 3.12 **Lecteur_joint_Etablissement(Numero_carte, Nom, Num_Etablissement_1, Num_Etablissement_2, Ville, Nom_Etablissement)**

Jointure des relations 'Lecteur_bis' et 'Etablissement' sur le champ 'Num _Etablissement'.

Numero _carte	Nom	Num_Etablis-sement_1	Num_Etablis-sement_2	Ville	Nom _Etablissement
1	Henri	1	1	Paris	Université Jussieu
2	Stanislas	2	2	Lyon	CHU Bron
3	Henriette	1	1	Paris	Université Jussieu

Le champ 'Num_Etablissement' y figure deux fois car une occurrence vient de la relation 'Lecteur' et l'autre de la relation 'Etablissement'. Afin de ne conserver qu'une valeur du champ 'Num_Etablissement', on utilise l'opération de **jointure naturelle** (voir figure 3.13).

Figure 3.13 **Lecteur_jointnat_Etablissement(Numero_carte, Nom, Num_Etablissement, Ville, Nom_Etablissement)**

Jointure naturelle des relations 'Lecteur_bis' et 'Etablissement' sur le champ 'Num _Etablissement'.

Numero_carte	Nom	Num _Etablissement	Ville	Nom _Etablissement
1	Henri	1	Paris	Université Jussieu
2	Stanislas	2	Lyon	CHU Bron
3	Henriette	1	Paris	Université Jussieu

On peut considérer l'opération de jointure comme une sélection sur le produit cartésien des deux relations. On ne conserve que les lignes dont le contenu du champ sur lequel s'effectue la jointure est égal. Les lignes grisées sont celles qui sont sélectionnées lors d'une jointure (voir figure 3.14).

Figure 3.14

Produit cartésien des relations 'Lecteur_bis' et 'Etablissement'.

Produit_cartesien_Etablissement(Numero_carte, Nom, Num_Etablissement_1, Num_Etablissement_2, Ville, Nom_Etablissement)

Numero _carte	Nom	Num_Etablis-sement_1	Num _Etablissement	Ville	Num_Etablis-sement_2
1	Henri	1	1	Paris	Université Jussieu
1	Henri	1	2	Lyon	CHU Bron
1	Henri	1	3	Nancy	Université Poincaré
1	Henri	1	4	Paris	Université Sorbonne
2	Stanislas	2	1	Paris	Université Jussieu
2	Stanislas	2	2	Lyon	CHU Bron
2	Stanislas	2	3	Nancy	Université Poincaré
2	Stanislas	2	4	Paris	Université Sorbonne
3	Henriette	1	1	Paris	Université Jussieu
3	Henriette	1	2	Lyon	CHU Bron
3	Henriette	1	3	Nancy	Université Poincaré
3	Henriette	1	4	Paris	Université Sorbonne

On verra au chapitre consacré à SQL que c'est l'une des manières d'écrire une jointure.

Jointure externe

La jointure externe n'est pas réellement une opération de base de l'algèbre relationnelle. Elle est cependant nécessaire pour pouvoir répondre plus facilement à des questions du type « Quels sont les établissements qui n'ont pas de lecteurs ? » Il s'agit d'une opération de jointure étendue qui inclut dans le résultat les lignes n'ayant pas de correspondance sur le contenu du champ de jointure. Voici le résultat de la jointure externe de la relation Etablissement avec la relation Lecteur sur le champ 'Num_Etablissement' (voir figure 3.15). On met en correspondance les valeurs du champ 'Num_Etablissement' de toutes les lignes de la relation 'Etablissement' avec celles de la relation 'Lecteur'.

Figure 3.15

Jointure externe des relations 'Lecteur_bis' et 'Établissement' sur le champ 'Num _Etablissement'.

Etablissement_jointext_ Lecteur(Num_Etablissement_1, Ville, Nom_Etablissement, Numero_carte, Nom, Num_Etablissement_2)

Num_Etablissement_1	Ville	Nom _Etablissement	Numero _carte	Nom	Num_Etablissement_2
1	Paris	Université Jussieu	1	Henri	1
2	Lyon	CHU Bron	2	Stanislas	2
3	Nancy	Université Poincaré	NULL	NULL	NULL
4	Paris	Université Sorbonne	NULL	NULL	NULL

Ici, les valeurs 3 et 4 du champ 'Num_Etablissement_1' provenant de la relation Etablissement n'ont pas de correspondance dans la relation 'Lecteur'. Les lignes sont tout de même incluses dans le résultat mais les champs provenant de la relation 'Lecteur' prennent la valeur 'NULL'. Cette valeur signifie que le champ ne contient pas de valeur.

Pour répondre à la question « Quels sont les établissements qui n'ont pas de lecteurs ? », il nous suffit de sélectionner les lignes qui contiennent la valeur 'NULL', par exemple dans le champ 'Numéro_carte'. Dans un second temps, on effectue une projection sur le champ 'Nom_Etablissement' (voir figure 3.16).

Figure 3.16

Sélection et projection sur la jointure externe des relations 'Lecteur_bis' et 'Etablissement' sur le champ 'Num _Etablissement'.

Etablissement_jointext_Lecteur_selproj(Nom_Etablissement)

Nom _Etablissement
Université Poincaré
Université Sorbonne

La jointure externe n'est pas une opération **symétrique**. L'opération inverse, c'est-à-dire la jointure externe de la relation 'Lecteur' avec la relation 'Etablissement' sur le champ 'Num_Etablissement' donne dans ce cas le même résultat qu'une jointure naturelle. En effet, toutes les valeurs du champ 'Num_Etablissement' de la relation 'Lecteur' ont une correspondance dans la relation 'Etablissement'. C'est ce que l'on souhaite puisque la relation 'Etablissement' fait office de relation de référence pour le champ 'Num_Etablissement' de la relation 'Lecteur'. L'opération de jointure externe peut être utilisée pour détecter ce type d'incohérence.

2.3 Calculs et agrégats

Une règle dans le domaine des bases de données est que tout ce qui peut se calculer ne doit pas être stocké. On évite ainsi la perte de place et l'incohérence qui peut en découler suite au stockage d'informations redondantes. Les opérations de l'algèbre relationnelle présentées dans les sections précédentes ne permettent pas de créer de nouveau champ par calcul à partir du contenu d'autres champs sans utiliser un langage de programmation. Des

fonctions de calculs ont été définies afin de répondre à ce besoin. Elles seront détaillées au chapitre sur « SQL ».

On considère la relation La_boutique(Num_facture, Article, Prix, Quantite) [voir figure 3.17].

Figure 3.17 **La_boutique(Num_Facture, Article, Prix, Quantite)**

Relation 'La_boutique'.

Num_Facture	Article	Prix	Quantite
101	Bananes	12	45
1034	Choux	5	129
2345	Riz	4	60
0987	Gazelle	15	48

On suppose que l'on veut exemple trouver le total pour chaque facture en multipliant le prix par la quantité. Il est alors possible d'ajouter un champ 'Total' dont le contenu sera calculé par l'expression 'Prix' * 'Quantité' (voir figure 3.18).

Figure 3.18 **La_boutique_total(Num_FactureArticle, Prix, Quantite, Total)**

Relation 'La_boutique' avec le champ calculé 'Total'.

Num_Facture	Article	Prix	Quantite	Total
101	Bananes	12	45	540
1034	Choux	5	129	645
2345	Riz	4	60	240
0987	Gazelle	15	48	720

Il est également possible d'effectuer des opérations statistiques globales d'un champ, comme les calculs du nombre d'enregistrements, de moyenne, de maximum, de minimum. On obtient dans ce cas une nouvelle relation réduite à une ligne et une colonne qui contient la valeur calculée. En effet, le résultat d'une opération sur une relation est toujours une relation (voir figure 3.19).

Figure 3.19 **La_boutique_moyenne(Moyenne_Prix)**

Moyenne des prix de la relation 'La_boutique'.

Moyenne_Prix
9

De même, les opérations de base de l'algèbre relationnelle ne permettent pas de répondre à des questions du type : « Combien trouve-t-on de personnes par ville ? » La solution consiste alors à regrouper les enregistrements qui contiennent les mêmes valeurs dans le champ 'Ville' : ce regroupement s'appelle un **agrégat**. On applique ensuite à chaque sous-relation ainsi constituée une opération statistique, dans notre cas l'opération de comptage (voir figure 3.20).

Figure 3.20 **Lecteur_parville(Ville, Nombre)**

Agrégat par ville de la relation 'La_boutique'.

Ville	Nombre
Bordeaux	1
Lyon	1

Ville	Nombre
Marseille	2
Nancy	3
Paris	5

3 Passage du modèle conceptuel au relationnel

On a exposé au chapitre sur l'analyse du monde réel deux formalismes différents – entité-association et UML – pour représenter le modèle conceptuel obtenu. Il existe des règles simples pour passer de ces modèles à un ensemble de relations qui sera utilisable directement dans le SGBD. Cet ensemble de relations s'appelle le **schéma relationnel** et constitue le **modèle logique des données**. On présente dans cette section les règles de transition qui permettent d'exprimer le schéma relationnel à partir du modèle conceptuel des données. Pour illustrer ces règles, on utilise la terminologie du modèle entité-association, mais leur expression est aisément adaptable au formalisme UML.

3.1 Règles générales

Deux grandes règles générales s'appliquent toujours, quel que soit le cas. Les exceptions que l'on aborde ensuite permettent simplement d'obtenir un schéma plus compact :

- Une entité devient une relation composée des champs de l'entité. La clé de cette relation est la même que celle de l'entité.
- Une association devient une relation composée des deux clés des entités associées. L'ensemble de ces deux clés constitue celle de la relation. Lorsqu'elle en possède, les champs de l'association forment les champs « non clés » de cette nouvelle relation.

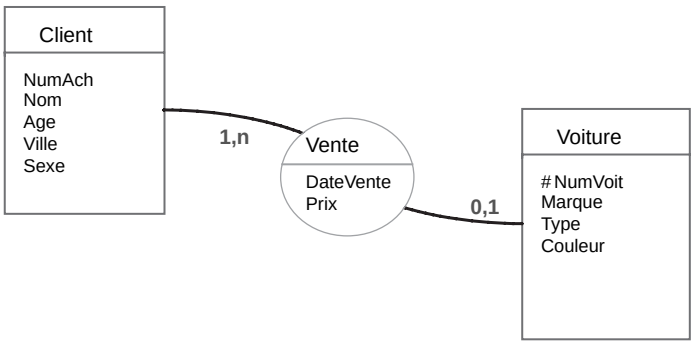


Figure 3.21
Modèle entité-association de la base de données 'casse'.

À partir du modèle entité-association de la base de données 'casse' élaboré au chapitre précédent (voir figure 3.21), on obtient trois relations en appliquant ces règles. Les entités 'voiture' et 'personne' deviennent les relations voiture(NumVoit, Marque, Type, Couleur) et personne(NumAch, Nom, Age, Ville, Sexe). L'association se transforme en relation vente(DateVent, Prix, NumAch, NumVoit).

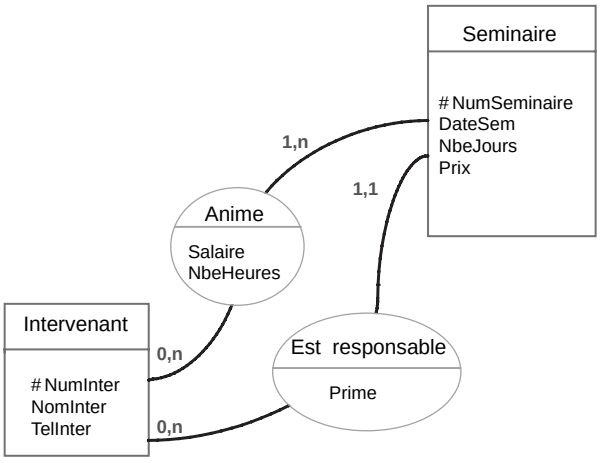
3.2 Cas particulier des associations avec une cardinalité de type '1-1'

Considérons le cas du modèle entité-association qui représente l'activité d'organisation de séminaires décrite très succinctement de la manière suivante :

- Un séminaire est animé par un ou plusieurs intervenants.
- Un séminaire ne possède qu'un seul responsable.

Figure 3.22

Modèle entité-association de la base de données 'animation de séminaire'.



Deux associations existent entre les deux entités 'Séminaires' et 'Intervenant' : 'Anime' et 'Est_responsable'. Les attributs utilisés pour les entités et les associations sont visibles sur le modèle entité-association (voir figure 3.22).

Si l'on applique les règles générales énoncées ci-dessus, on obtient quatre relations :

- Intervenant(NumInter, NomInter, TelInter)
- Seminaire(NumSem, DateSem, Prix, NbeJour)
- Anime(NumInter, NumSem, NbeHeure, SalaireHor)
- Est_responsable(NumInter, NumSem, Prime)

L'une des cardinalités de l'association 'Est_responsable' est de type '1-1', car un séminaire possède un responsable et un seul. De cette dernière propriété, on peut en déduire que la clé de la relation 'Est_responsable' n'est pas minimale. En effet, pour la relation 'Est_responsable' l'identifiant du séminaire détermine celui du responsable. En d'autres termes, on identifie une dépendance fonctionnelle entre les champs 'NumSem' et 'NumInter' pour la relation 'Est_responsable'. On choisit le champ 'NumSem' comme clé de la relation 'Est_responsable'.

La relation 'Est_responsable' devient ainsi Est_responsable(NumSem, NumInter, Prime).

Les relations 'Est_responsable' et 'Seminaire' ont alors la même clé, et il est possible de les regrouper. On obtient la relation Seminaire_Res(NumSem, DateSem, Prix, NbeJour, NumInter, Prime). Il s'est produit une sorte de « glissement » des éléments de l'association 'Est_responsable' vers l'entité 'Seminaire'.

On peut en déduire une règle complémentaire des règles générales précédentes :

- Lorsque l'association entre deux entités comprend une cardinalité de type '1-1', on ne crée pas de relation pour l'association. Les champs de l'association seront intégrés à la

relation créée pour l'entité qui est associée avec la cardinalité '1-1'. La clé de cette relation est toujours celle de l'entité.

Remarque

L'un des intérêts d'UML est de disposer de logiciels capables, à partir d'un modèle conceptuel exprimé en UML, de déduire automatiquement les relations en utilisant les règles énoncées ici. D'autres règles apparaîtraient dans le cas de l'utilisation des extensions objet que propose UML ; cependant ces possibilités dépassent le cadre de cet ouvrage.

4 Normalisation

Le modèle relationnel procure des outils destinés à tester la qualité et la cohérence des relations dans un schéma relationnel créé à l'étape précédente. Cette étape, appelée **normalisation**, permettra de vérifier certaines propriétés des relations et le cas échéant de les transformer.

On aborde dans cette section les trois premières formes normales qui suffisent dans la plupart des cas et qui permettent une décomposition du schéma relationnel sans perte d'information. La forme normale de Boyce-Codd qui détecte d'autres incohérences, mais propose une décomposition avec perte d'informations de dépendance, est présentée à la fin de la section. Il existe d'autres formes normales – la quatrième et la cinquième –, qui ne sont pas présentées dans cet ouvrage : ces dernières sont parfois difficiles à appréhender et les trois premières formes normales suffisent en général pour obtenir un schéma relationnel de qualité.

La normalisation d'un schéma relationnel suggère une autre méthode pour obtenir un ensemble de relations. On part d'une relation unique qui contient tous les champs, que l'on appelle la **relation universelle**. À l'aide des décompositions proposées par la mise en forme normale et du graphe des dépendances fonctionnelles des champs de cette relation, on parvient par raffinements successifs à un ensemble de relations normalisées. Cette méthode de la relation universelle est toutefois assez difficile à manipuler dès que l'on dépasse une taille critique du nombre de champs.

4.1 Première forme normale

La première forme normale s'intéresse au contenu des champs. Elle interdit la présence, appelée **multivaluation,** de plusieurs valeurs dans un même champ d'une relation. En effet, la notion de dépendance fonctionnelle entre les champs ne peut plus être vérifiée s'ils possèdent plusieurs valeurs. Elle s'exprime de la manière suivante :

Tout champ contient une valeur atomique.

Comment passer en première forme normale ?

La relation suivante n'est pas en première forme normale ; le champ 'Auteurs' contient plusieurs valeurs (voir figure 3.23).

Figure 3.23 **Publication(NumPubli, Titre, Auteurs)**

Relation 'Publication' avec un champ multivalué.

NumPubli	Titre	Auteurs
13490	Le vin et l'avenir	Jean Lasso, Hubert De la Tuque, Stanislas Wilski
21322	Bière et progrès social	Aristide Salem, Jean Lasso, Salome Dupont
45333	Le champagne et la France	Penelope Light, Vanessa Martinez, Salome Dupont

Une solution pour résoudre ce problème est de décomposer le champ 'Auteurs' en 'Auteur_1', 'Auteur_2', 'Auteur_3', 'Auteur_4', etc. Ainsi, la relation est bien en première forme normale. L'ennui est que l'on ne sait pas à l'avance le nombre d'auteurs que peut posséder une publication, et le problème consiste donc à savoir combien de champs ajouter.

La solution plus correcte, mais également plus lourde à mettre en œuvre, est de décomposer cette relation en trois relations : Publication(NumPubli, Titre), Auteur(NumAuteur, Nom, Prenom) et EstEcrite(NumPubli, NumAuteur). Ces trois relations sont la représentation de la réalité « une publication est écrite par des auteurs ». Elle se modélise par deux entités 'Publication' et 'Auteur' reliées par l'association 'EstEcrite', comme on l'a vu au chapitre 2, « Analyse du monde réel ».

On obtient alors le résultat suivant (voir figure 3.24).

Figure 3.24 **Publication(NumPubli, Titre)**

Décomposition de la relation 'Publication' en trois relations.

NumPubli	Titre
13490	*Le vin et l'avenir*
21322	Bière et progrès social
45333	Le champagne et la France

Auteur(NumAuteur, Nom, Prenom)

NumAuteur	Nom	Prenom
1	Lasso	Jean
2	De la Tuque	Hubert
3	Wilski	Stanislas
4	Salem	Aristide
5	Dupont	Salome
6	Light	Penelope
7	Martinez	Vanessa

EstEcrite(NumPubli, NumAuteur)

NumPubli	NumAuteur
13490	1
13490	2
13490	3
21322	4
21322	1
21322	5
45333	6
45333	7
45333	5

On doit alors effectuer des jointures sur les différentes relations afin de reconstituer l'information dans son intégralité. Cette décomposition en trois relations se fait sans perte d'information.

4.2 Deuxième forme normale

Bien évidemment, une relation doit déjà se trouver en première forme normale pour être en deuxième forme normale. Cette dernière recherche la redondance d'information dans une relation. Elle interdit les dépendances fonctionnelles possibles entre les champs qui composent la clé et les autres champs. On peut l'exprimer de la manière suivante :

La relation est en première forme normale.

Tout champ qui n'appartient pas à la clé ne dépend pas d'une partie de la clé.

Comment passer en deuxième forme normale ?

La solution consiste à décomposer la relation en deux relations. La nouvelle relation créée a pour clé la partie de la clé dont dépendent les autres champs qui constituent ses champs.

On considère la relation 'Produit' suivante (voir figure 3.25).

Figure 3.25 Relation 'Produit'.

Produit(Article, Fournisseur, Adresse, Prix)

Article	Fournisseur	Adresse	Prix
Marteau	SOGENO	Paris	5
Tournevis	ARTIFACT	Lille	10
Tournevis	SOGENO	Paris	23
Pince	LEMEL	Paris	34
Mètre	ARTIFACT	Lille	24

La clé est constituée des champs 'Article' et 'Fournisseur'. Or, il y a une relation de dépendance entre le champ 'Fournisseur', qui est une partie de la clé, et le champ 'Adresse'.

On décompose alors la relation pour éliminer la redondance ainsi créée. La nouvelle relation

aura pour clé la partie de la clé de la relation d'origine dont dépendent fonctionnellement les autres champs. Dans cet exemple, il s'agit du champ 'Fournisseur'. Les autres champs dépendants constituent le reste de la relation. Il s'agit ici du champ 'Adresse'.

On obtient alors le résultat suivant (voir figure 3.26).

Figure 3.26 **Décomposition de la relation 'Produit' pour passer en deuxième forme normale.**

Produit(Article, Fournisseur, Prix)

Article	Fournisseur	Prix
Marteau	SOGENO	5
Tournevis	ARTIFACT	10
Tournevis	SOGENO	23
Pince	LEMEL	34
Mètre	ARTIFACT	24

Fournisseur(<u>Fournisseur</u>, Adresse)

Fournisseur	Adresse
SOGENO	Paris
ARTIFACT	Lille
LEMEL	Paris

Comme précédemment, il est nécessaire de faire une jointure pour reconstituer l'information. La décomposition en deux relations se fait sans perte d'information.

Remarque

Si la clé d'une relation est atomique, c'est-à-dire composée d'un seul champ, elle est naturellement en deuxième forme normale.

4.3 Troisième forme normale

La troisième forme normale recherche également la redondance d'information dans une relation. On cherche s'il existe une dépendance entre deux champs qui ne font pas partie d'une clé. Si c'est le cas, on se trouve dans la situation où un champ dépend d'un autre champ qui dépend lui même d'une clé. La clé considérée peut être primaire ou secondaire. La troisième forme normale interdit donc les dépendances fonctionnelles dites « **transitives** » entre les champs. Elle s'exprime de la manière suivante :

La relation est en deuxième forme normale (donc en première forme normale).

Tout champ n'appartenant pas à une clé ne dépend pas d'un autre champ non clé.

Comment passer en troisième forme normale ?

La solution est également de décomposer la relation de départ en deux relations. La nouvelle relation créée a pour clé le champ dont dépendent les autres champs qui constituent ainsi la dépendance transitive.

On considère la relation suivante (voir figure 3.27).

Figure 3.27 **Baladeur(NumBal, Marque, Type, Couleur)**

Relation 'Baladeur'.

NumBal	Marque	Type	Couleur
12	Apple	Ipod	Blanc
43	Creative	Zen	Noir
23	Apple	Ipod	Noir
29	Creative	Zen	Gris
34	Sony	MZ-RH910	Rouge

La clé de cette relation est un numéro, 'NumBal', car il peut y avoir dans notre stock plusieurs baladeurs de même marque, de même type et de même couleur. Les marques déposent les noms des objets qu'elles fabriquent de façon à les identifier sur le marché. Il existe donc une dépendance fonctionnelle entre le champ 'Type' (qui n'appartient pas à la clé) et le champ 'Marque' (qui n'appartient pas à la clé). On décompose la relation en en créant une nouvelle qui a pour clé le champ dont dépendent les autres champs constituant la dépendance transitive. Il s'agit dans ce cas du champ 'Type'. Les autres champs de la nouvelle relation sont composés des champs qui en dépendent fonctionnellement : ici, le champ 'Marque' (voir figure 3.28).

Figure 3.28 **Baladeur(NumBal, Type, Couleur)**

Décomposition de la relation 'Baladeur' pour passer en troisième forme normale.

NumBal	Type	Couleur
12	Ipod	Blanc
43	Zen	Noir
23	Ipod	Noir
29	Zen	Gris
34	MZ-RH910	Rouge

Baladeur_type(Type, Marque)

Type	Marque
Ipod	Apple
MZ-RH910	Sony
Zen	Creative

Comme précédemment, il est nécessaire de faire une jointure pour reconstituer l'information dans son intégralité. La décomposition en deux relations se fait sans perte d'information.

Remarque

Les deuxième et troisième formes normales traitent des problèmes différents. L'ordre dans lequel on les considère pour la normalisation, mise en deuxième forme puis en troisième forme normale, est plutôt lié à l'historique qu'à une nécessité réelle.

4.4 Forme normale de Boyce-Codd

La forme normale de Boyce-Codd traite un cas un peu différent de ceux de la deuxième et troisième forme normale. Il s'agit du cas où une partie d'une clé dépend d'un champ. Comme pour la troisième forme normale, la clé considérée peut être une clé primaire ou secondaire. Une relation en troisième forme normale n'est pas toujours en forme « Boyce-Codd », mais l'inverse est toujours vrai.

Tout champ appartenant à une clé ne dépend pas d'un autre champ non clé.

Comment passer en forme normale de Boyce-Codd ?

Lors de la décomposition de la relation en deux relations, plusieurs choix sont envisageables compte tenu des liens de dépendances multiples entre les différents champs. On choisit la décomposition qui permet de reconstituer strictement l'information d'origine sans générer de données supplémentaires. La perte d'une dépendance fonctionnelle est fréquente lors de cette opération. La relation créée a pour clé la partie de celle-ci dont dépendent les autres champs constituant la dépendance.

On considère la relation suivante (voir figure 3.29).

Figure 3.29 **Dictaphone(Marque, Produit, Prix, Couleur)**

Relation 'Dictaphone'.

Marque	Produit	Prix	Couleur
Philips	LD 1024	49	Blanc
Olympus	VN 1664	49	Noir
Philips	LD 5647 H	59	Blanc
ImaginR	VN 1664	69	Gris
Olympus	VN 234 PC	79	Rouge

La clé de cette relation est constituée par les champs 'Marque' et 'Produit'. En effet, un produit est fabriqué sous licence par la société ImaginR et a donc le même nom que celui proposé par la société Olympus. Il est alors nécessaire d'utiliser les deux champs pour constituer la clé. Pour se démarquer les unes des autres, les sociétés utilisent des couleurs personnalisées destinées à identifier la marque : Philips, le blanc et l'orange ; Olympus, le rouge et le noir ; ImaginR, le gris. On a donc une relation de dépendance entre ces deux champs. La relation est en troisième forme normale, mais elle n'est pas en forme de Boyce-Codd.

Plusieurs décompositions sont possibles : par exemple Dictaphone(Marque, Type, Prix) et Marque_coul(Couleur, Marque). Mais cette décomposition génère des tuples non désirés au moment de la jointure. Quant à la décomposition Dictaphone(Type, Prix, Couleur) et Marque_coul(Couleur, Marque), elle permet de reconstituer l'information de départ par une jointure sur le champ 'Couleur' (voir figure 3.30).

Figure 3.30 **Dictaphone(Produit, Prix, Couleur)**

Relation 'Dictaphone' décomposée pour passer en forme de Boyce-Codd.

Produit	Prix	Couleur
LD 1024	49	Blanc
VN 1664	49	Noir
LD 5647 H	59	Blanc

Produit	Prix	Couleur
VN 1664	69	Gris
VN 234 PC	79	Rouge

Marque_coul(Couleur, Marque)

Couleur	Marque
Blanc	Philips
Noir	Olympus
Gris	ImaginR
Rouge	Olympus

On a perdu la dépendance de 'Couleur' par rapport à 'Marque', 'Produit'.

5 Logique du premier ordre et base de données

Dans cette section, le fondement logique de l'approche relationnelle est brièvement abordé pour présenter une méthode d'interrogation : les QBE (*Query By Example*). La logique du premier ordre a pour but de formaliser le raisonnement naturel en considérant la déduction comme le résultat d'un calcul. Elle a fait l'objet de recherches intenses depuis l'Antiquité : Aristote, Frege et Gödel pour n'en citer que quelques-uns sont parmi les plus prestigieux à s'être penchés sur la question. Les principes de la logique du premier ordre, dans une version restreinte que l'on appelle **le calcul des propositions**, ont trouvé de nombreuses applications en informatique. Après quelques rappels sur le formalisme de la logique, on aborde ses liens avec l'approche relationnelle en base de données et l'on présente la méthode d'interrogation par QBE.

5.1 Rappels sur la logique du premier ordre

Le formalisme de la logique du premier ordre permet d'exprimer des phrases dans un langage non ambigu. Il offre la possibilité, par des règles de dérivation, ou réécritures, de déduire d'autres phrases. L'élément fondamental de la logique du premier ordre est le **prédicat** qui exprime le lien entre différents éléments. La validité de ce prédicat est généralement restreinte à un ensemble de valeurs que l'on nomme **domaine du discours**.

```
mange(x,y) sera par exemple vrai pour les valeurs du couple (x,y) suivantes
(homme,navet)
(lapin,carotte)
(homme,lapin).
```

On utilise classiquement des variables (x,y dans notre exemple) et des constantes dans les expressions. Les prédicats peuvent être associés par des connecteurs logiques : $\wedge$ (et), $\vee$ (ou), $\neg$ (non), $\Rightarrow$ (implique). Le domaine de discours des variables s'exprime à partir des quantificateurs $\forall$ (quel que soit) et $\exists$ (il existe). Ce formalisme simple permet d'exprimer des propriétés comme celle-ci : « si x est mangé par y ou y est mangé par x, ils appartiennent tous deux à la même chaîne alimentaire ».

```
∀ x ∃ y mange(x,y) ∨ mange(y,x) ⇒ meme_chaine_alimentaire(x,y)
```

Cette formule peut se réécrire en utilisant des règles de déduction dans le but d'aboutir à un ensemble de clauses plus aisées à vérifier : c'est sur ce principe que fonctionne la démonstration automatique de théorèmes. La différence entre un (bon) mathématicien et un programme est évidemment que le mathématicien va choisir d'emblée la série de règles adaptée pour parvenir plus rapidement au résultat.

Voici une possibilité de réécriture de la formule précédente.

a ⇒ b donne ¬a ∨ b
∀ x ∃ y (¬(mange(x,y) ∨ mange(y,x)) ∨ (meme_chaine_alimentaire(x,y)))

¬ (a ∨ b) donne ¬a ∧ ¬b
∀ x ∃ y (¬mange(x,y) ∧ ¬ ? mange(y,x)) ∨ (meme_chaine_alimentaire(x,y)))

(¬a ∧ ¬b) ∨ c donne (¬a ∨ c) ∧ (¬b ∨ c)
∀ x ∃ y ((¬mange(x,y) ∨ meme_chaine_alimentaire(x,y)) ∨ (¬(mange(y,x) ∨ meme_chaine_alimentaire(x,y)))

¬a ∨ b donne a ⇒ b
∀ x ∃ y ((mange(x,y) ⇒ meme_chaine_alimentaire(x,y)) ∨ (mange(y,x) ⇒ meme_chaine_alimentaire(x,y))

∀ x ∃ y signifie que y est défini en fonction de x, on remplace y par F(x)

On obtient ainsi deux clauses reliées par ∨, on l'appelle clause de Horn qui représente la fin de la réécriture :

```
mange(x,F(x)) ⇒ meme_chaine_alimentaire(x,F(x)))
mange(F(x),x) ⇒ meme_chaine_alimentaire(x, F(x))
```

Dans notre cas, on peut s'assurer assez facilement que ces deux implications sont toujours vérifiées pour le domaine du discours.

5.2 Lien avec les bases de données

Le lien avec ce qui a été énoncé précédemment sur le modèle de bases de données relationnelles est le suivant :

- Le prédicat correspond à la relation.
- Le domaine du discours correspond au contenu de la relation.

Ce formalisme permet d'exprimer les questions, naturellement ambiguës du langage parlé, par une formule. Cette formule est réécrite en utilisant les règles vues précédemment pour arriver à un ensemble de clauses simples. On constitue ensuite des sous-ensembles de tuples de la relation pour lesquels les clauses réécrites sont vérifiées : ces tuples représentent la réponse à la question posée. Voici quelques exemples de questions exprimées en utilisant ce formalisme. On obtient ainsi des sortes de « patrons » de recherche permettant de caractériser les tuples qui sont solutions.

Dans l'exemple général (voir section 3.1), le schéma de la relation 'voiture' est voiture(NumVoit, Marque, Type, Couleur) et le schéma de la relation 'vente' est vente(DateVent, Prix, NumAch, NumVoit).

L'expression « afficher la liste des marques et des types de voitures », ou projection des champs 'Marques' et 'Types 'de la relation 'voiture', peut s'écrire sous la forme :

```
{ (m,t) | voiture(_,m,t,_) }
```

Cela signifie que les valeurs des champs 'Marque' et 'Type' des tuples de la relation 'voiture' sont représentées par les variables 'm' et 't'. Le signe '_' représente n'importe quelle valeur des autres champs 'NumVoit' et de 'Couleur'.

L'expression « afficher le type des voitures de couleur rouge », qui est une projection et une sélection sur la relation 'voiture', peut s'écrire sous la forme :

```
{ (t) | voiture(_,_,t,'rouge') }
```

Cette fois, on utilise une constante, 'rouge', dans l'expression pour fixer la valeur d'un champ et la variable, 't', pour les valeurs du champ recherchées.

On peut exprimer l'appartenance d'un champ à un ensemble par un critère. L'expression « afficher les prix de vente supérieurs à 10 000 », qui est une projection et une sélection sur la relation 'vente', peut s'écrire sous la forme :

```
{ (p) | vente(_,p,_,_) ∧ p > 10000 }
```

Enfin, la jointure entre deux relations est également possible. L'expression « afficher le type des voitures vendues et leur prix », qui met en jeu deux relations, peut s'écrire sous la forme :

```
{ (t,p) | ∃ nv voiture(nv,_,t,_) ∨ vente(_,p,_ ,nv)}
```

Il suffit d'utiliser la même variable dans les deux relations : ici 'nv', pour le champ 'NumVoit' qui sert à effectuer le lien. Le quantificateur '∃' permettra de ne sélectionner que les tuples dans les deux relations pour lesquels les valeurs de 'NumVoit' sont identiques, ce qui est exactement la définition d'une jointure (équijointure).

Le formalisme de la logique du premier ordre permet d'exprimer toutes les opérations relationnelles vues précédemment : c'est normal puisqu'il s'agit de la base de l'approche relationnelle. On peut alors considérer un SGBD comme un outil de démonstration de théorèmes, tels que l'on peut en rencontrer en intelligence artificielle, qui agirait sur un ensemble très restreint de règles.

5.3 Interrogation par QBE (*Query By Example*)

Lors de la phase de développement du prototype de SGBD basé sur l'approche relationnelle dans les laboratoires d'IBM, un premier sous-produit a été conçu : le langage d'interrogation SEQUEL, puis SQL. Une autre méthode d'interrogation a été développée à cette occasion : les **QBE** (*Query By Example*). L'idée est de disposer d'un mode d'interrogation d'une base de données sans connaître de langage et en utilisant le formalisme vu à la section précédente, présenté sous une forme graphique. Comme son nom l'indique, on va utiliser des exemples pour définir les questions. La relation est représentée sous la forme d'un tableau :

La projection d'un champ se fait en cochant la case correspondant au nom de la colonne (voir figure 3.31).

Figure 3.31

Projection du champ 'Marque' dans un QBE.

Afficher la liste des marques et des types de voitures :

```
{ (m,t) | voiture(_,m,t,_) }
```

☐ NumVoit	☒ Marque	☒ Type	☐ Couleur

- Les critères de sélection se font par des valeurs exemples (voir figures 3.32 et 3.33).

Figure 3.32 **Afficher le type des voitures de couleur rouge :**

Sélection sur un contenu et projection dans un QBE.

```
{ (t) | voiture(_,_,t,'rouge') }
```

☐ NumVoit	☐ Marque	☒ Type	☐ Couleur Rouge

Figure 3.33 **Afficher les prix de vente supérieurs à 10 000 :**

Sélection sur un critère et projection dans un QBE.

```
{ (p) | vente(_,p,_,_) ∧ p > 10000 }
```

☐ DateVent	☒ Prix	☐ NumAch	☐ NumVoit
	> 10 000		

- Les conditions situées sur la même ligne sont reliées par un « et » (voir figure 3.34).

Figure 3.34 **Afficher les prix de vente supérieurs à 10 000 et dont la date de vente a eu lieu après le 1er janvier 1997 :**

Sélection multicritère obligatoire et projection dans un QBE.

```
{ (p) | vente(d,p,_,_) ∧ (p > 10000 ∧ d > '01/01/1997')}
```

☐ DateVent	☒ Prix	☐ NumAch	☐ NumVoit
> '01/01/1997'	> 10 000		

- Les conditions situées sur deux lignes différentes sont reliées par un « ou » (voir figure 3.35).

Figure 3.35 **Afficher les prix de vente supérieurs à 10 000 ou dont la date de vente a eu lieu après le 1er janvier 1997 :**

Sélection multicritère optionnel et projection dans un QBE.

```
{ (p) | vente(d,p,_,_) ∧ (p > 10000 ∨ d > '01/01/1997')}
```

☐ DateVent	☒ Prix	☐ NumAch	☐ NumVoit
> '01/01/1997'	> 10 000		

- La jointure entre deux relations se fait en utilisant une variable, exactement comme dans une formule de la logique de premier ordre (voir figure 3.36).

Figure 3.36 **Afficher le type des voitures vendues et leur prix :**

Jointure et projection dans un QBE.

```
{ (t,p) | ∃ nv voiture(nv,_,t,_) ∨ vente(_,p,_ ,nv)}
```

☐ NumVoit	☐ Marque	☒ Type	☐ Couleur
Nv			

☐ DateVent	☒ Prix	☐ NumAch	☐ NumVoit
			Nv

Cette méthode d'interrogation simple et efficace est encore proposée par de nombreux SGBD.

Résumé

L'approche relationnelle modélise les faits de la vie réelle par des tuples, qui sont des ensembles de valeurs de différents champs (ou attributs) : (réfrigérateur, 2003, rouge) est un tuple qui représente des valeurs des champs 'Objet', 'Année', 'Couleur' liées dans le monde réel. L'ensemble des tuples s'appelle une relation. Il s'agit du concept de base qui sera manipulé par l'approche relationnelle et peut être représenté sous la forme d'une table.

Pour identifier un tuple, on utilise le contenu d'un ou de plusieurs champs que l'on nommera la **clé** d'une relation. Cette dernière est établie en utilisant le concept de dépendance fonctionnelle entre les différents champs. La clé est constituée par le plus petit ensemble de champs dont dépendent fonctionnellement les autres champs. Si plusieurs clés sont possibles, on parle de clés candidates. La clé choisie sera nommée la clé primaire.

Les opérations de manipulation de ces relations peuvent être regroupées en trois catégories :

- **Les opérations du monde ensembliste.** Produit cartésien, intersection, union et différence.
- **Les opérations spécifiques relationnelles.** Projection, sélection (ou restriction) et jointure. On a présenté également la jointure externe qui permet de répondre à des questions spécifiques même s'il ne s'agit pas d'une opération de base du relationnel.
- **Les opérations et les fonctions de calcul.** Elles ne constituent pas réellement des opérations du monde relationnel mais sont utiles pour effectuer des calculs sans recourir à un langage de programmation.

La cohérence des données contenues dans les relations est améliorée par la définition de contraintes que l'on appliquera sur les relations au moment de leur création. Ces contraintes expriment essentiellement les conditions d'appartenance à un ensemble que l'on nomme le domaine du champ. On peut décrire cet ensemble de plusieurs manières :

- Une liste exhaustive de valeurs, comme celles des jours de la semaine : « lundi », « mardi », etc.
- Le respect de propriétés, comme : « l'âge doit être compris entre 7 et 77 ans ».
- L'utilisation des valeurs d'un champ comprises dans une autre relation de référence. On parle de clé étrangère.

La mise en œuvre de ces contraintes est assurée par le SGBD en utilisant le *Langage de Définition de Données* (LDD). Après avoir présenté les concepts de relation (ou de table) et les outils qui permettent de les manipuler, on a décrit les règles qui conduisent du modèle conceptuel présenté au chapitre précédent à un ensemble de relations constituant le modèle logique de la base de données. Les liaisons entre les relations, qui expriment les liens de sens dans la réalité, seront établies dynamiquement par l'opération fondamentale de l'approche relationnelle qui est la jointure.

On évalue ensuite la qualité de ces relations en vérifiant leur conformité par rapport à des propriétés que l'on appelle les formes normales. Ces propriétés visent essentiellement à détecter la redondance et la cohérence des données dans les relations. On a présenté dans ce chapitre les quatre formes normales les plus courantes :

- la première forme normale qui interdit les champs multivalués ;
- la deuxième forme normale qui détecte une relation de dépendance entre une partie de la clé et un champ ;

- la troisième forme normale qui détecte une dépendance « transitive » entre une clé et un champ, c'est-à-dire qu'un champ non-clé dépend d'un autre champ non-clé qui dépend lui même de la clé ;
- la forme normale de Boyce-Codd qui détecte la relation de dépendance entre un champ non-clé et une partie d'une clé ; c'est une extension de la troisième forme normale qui est plus restrictive.

La normalisation est une méthode de réorganisation qui consiste à décomposer une relation pour la rendre conforme aux formes normales. La recomposition des données se fait alors par une opération de jointure qui peut se révéler coûteuse en ressources. C'est pour cette raison de performance que certaines relations sont parfois laissées en forme non normalisée. Les deux étapes précédentes de passage du modèle conceptuel au schéma relationnel et de normalisation peuvent être quasiment automatisées. Enfin, on a abordé les bases du fondement logique de l'approche relationnelle. L'objectif est de présenter une méthode d'interrogation intuitive et graphique d'une base de données qui en découle : les QBE ou « interrogation par l'exemple ».

Exercices

EXERCICE 1 RELATION, DEGRÉ, CARDINALITÉ

Énoncé

On considère deux relations. La première *Garage* est de degré 7 et de cardinalité 3. La seconde *Film* est de degré 2 et de cardinalité 15. Quels sont le degré et la cardinalité du produit cartésien de *Garage* par *Film ?* Quels sont le degré et la cardinalité du produit cartésien de *Film* par *Garage* ?

Solution

Comme le produit cartésien est une combinaison de tous les tuples des deux relations, le nombre de tuples sera le produit du nombre de tuples des deux relations. Tous les champs sont intégrés dans le résultat en une sorte de juxtaposition : le nombre de champs du produit cartésien est égal à la somme du nombre de champs des deux relations. Enfin, le produit cartésien est une opération symétrique ; le résultat sera le même pour les deux questions.

La relation *Garage* possède 3 tuples de 7 champs et la relation *Film* possède 15 tuples de 2 champs. Le résultat est une relation qui possède 45 tuples et 9 champs. On peut également dire que le résultat est une table qui possède 45 lignes et 9 colonnes.

EXERCICE 2 CLÉ D'UNE RELATION

Énoncé

Quelle est la clé de cette relation (voir figure 3.37) ?

Film(Prix, Format, Type, Nombre)

Figure 3.37

Relation 'Film'.

Prix	Format	Type	Nombre
12	4 :3	Couleur	3
4	16 :9	Noir/Blanc	1
12	16 :9	Couleur	1664
35	4 :3	Noir/Blanc	890
12	16 :9	Noir/Blanc	1

Solution

Si l'on considère simplement le contenu de la relation, il n'y a pas de clé constituée par un seul champ. Il semble que la combinaison de deux champs, 'Prix' et 'Nombre' est une clé et que c'est la seule qui contient deux champs.

On pourrait trouver des combinaisons avec trois champs qui fonctionnent : par exemple 'Prix' et 'Format' et 'Couleur'. Ce serait alors une clé candidate, mais on choisit celle qui est la plus 'atomique' possible. On peut s'interroger en revanche sur le bien-fondé d'une clé constituée des deux champs 'Prix' et 'Nombre' si l'on considère leur sens dans le monde réel. Il semble évident que l'on puisse avoir deux films différents avec le même prix et le même nombre.

Comment choisir la clé dans ce cas ? On ne peut donc en général pas choisir une clé sans tenir compte des dépendances fonctionnelles entre les champs. Même si les données présentes dans

la relation semblent confirmer qu'il s'agit bien d'une clé, il n'y a pas ici de dépendance entre les champs 'Prix' et 'Nombre'. Sauf si le commanditaire de la base de données vous affirme le contraire pour son cas particulier. Sans faire une analyse exhaustive des dépendances entre tous les champs, il semble qu'il n'y a pas dans cette relation de « bons » candidats pour identifier un tuple. Une solution est d'ajouter un champ spécifique ; c'est ce que proposent la plupart des SGBD lorsque vous ne définissez pas de clé pour une relation.

EXERCICE 3 CONTRAINTES D'INTÉGRITÉ

Énoncé

On considère la relation *Film(Prix, Format, Type, Nombre)* de l'exemple précédent. Proposez des contraintes d'intégrité pour chaque champ. On suppose que l'on ajoute un champ 'Numéro_Film' qui correspond à son identifiant dans une relation descriptive qui est un catalogue de films. Que proposez-vous comme contrainte pour ce champ ?

Solution

Pour les champs de type numérique comme 'Prix' et 'Nombre', on peut proposer des limites d'appartenance à un intervalle. Le prix doit être compris entre 0 et 1 000, et le nombre entre 0 et 10 000.

Pour les champs de type caractère comme 'Format' et 'Couleur', il semble que les valeurs puissent être incluses dans des ensembles énumérés. Le format peut être compris dans l'ensemble (3 :4,16 :9) et la couleur dans l'ensemble ('Couleur','Noir/Blanc') .

Le contenu du champ 'Numéro_Film' peut être défini par rapport au contenu du champ correspondant dans la relation 'catalogue'. Cela revient à imposer la contrainte suivante : on n'entre pas de numéro de films qui ne se trouveraient pas dans le catalogue. Dans tous les cas, ces valeurs sont à déterminer avec les usagers de la base de données.

EXERCICE 4 OPÉRATION ENSEMBLISTE

Énoncé

Exprimez l'intersection entre deux relations à partir des opérations d'union et de différence. Donnez-en une illustration avec ses deux relations (voir figure 3.38).

Figure 3.38

Relations 'ma_cuisine' et 'sa_cuisine'.

ma_cuisine(Appareil, Couleur)

Appareil	Couleur
Réfrigérateur	rouge
Robot	mauve
Cuisinière	jaune

sa_cuisine(Appareil, Couleur)

Appareil	Couleur
Réfrigérateur	mauve
Cuisinière	jaune
Hotte	bleue

Solution

Pour réaliser ces opérations, il faut que les relations possèdent le même schéma. L'intersection entre deux ensembles peut se concevoir de la manière suivante en utilisant l'opé-

rateur de différence : *ma_cuisine ∩ sa_cuisine = ma_cuisine – (ma_cuisine - sa_cuisine)* [voir figure 3.39].

Figure 3.39 **ma_cuisine - sa_cuisine**

Opération sur les relations 'ma_cuisine' et 'sa_cuisine'.

Appareil	Couleur
Réfrigérateur	rouge
Robot	mauve

ma_cuisine – (ma_cuisine - sa_cuisine)

Appareil	Couleur
Cuisinière	jaune

On obtient bien le résultat de l'intersection de *ma_cuisine ∩ sa_cuisine*. Vérifiez par la même méthode que *sa_cuisine ∩ ma_cuisine* peut s'écrire *sa_cuisine – (sa_cuisine - ma_cuisine)* et que l'opération est symétrique bien que l'opération de différence ne le soit pas.

EXERCICE 5 PROJECTION

Énoncé

Trouver le prix et le type de tous les films de la relation 'Film' vue précédemment. Peut-on en déduire que 'Prix' & 'Type' est une clé candidate, c'est-à-dire que toute combinaison des valeurs du prix et du type d'un film permet d'identifier un film ?

Solution

Il s'agit simplement d'une projection sur les champs (voir figure 3.40).

Figure 3.40 **Film_proj(Prix, Type)**

Projection de la relation 'Film'.

Prix	Type
12	Couleur
4	Noir/Blanc
12	Couleur
35	Noir/Blanc
12	Noir/Blanc

Cette projection permet de mettre en évidence que 'Prix' et 'Type' n'est pas une clé candidate, car il y a un doublon : le tuple (12, Noir/Blanc).

EXERCICE 6 RESTRICTION

Énoncé

Donnez le prix des films de la base films en « Noir/Blanc ». Quels sont le degré et la cardinalité de la relation obtenue ? Est-il possible de calculer ces valeurs à l'avance, comme on le fait pour un produit cartésien ?

Solution

Il s'agit de réaliser une restriction sur les tuples dont le champ 'Type' contient « Noir/Blanc » ; on fait ensuite une projection sur le champ 'Prix' (voir figure 3.41).

Figure 3.41

Restriction et projection de la relation 'Film'.

Prix	Format	Type	Nombre
12	4 :3	Couleur	3
4	16 :9	Noir/Blanc	1
12	16 :9	Couleur	1664
35	4 :3	Noir/Blanc	890
12	16 :9	Noir/Blanc	1

On obtient une relation de degré 1 et de cardinalité 2 (voir figure 3.42).

Figure 3.42

Résultat de la restriction et de la projection de la relation 'Film'.

Film_NB(Prix)

Prix
4
35

On peut calculer le degré facilement, puisqu'il s'agit du nombre de champs sur lesquels on fera la projection. Pour la cardinalité qui représente le nombre de lignes du résultat, on ne peut la calculer à l'avance puisqu'elle va dépendre du contenu du champ 'Type' des tuples de la relation.

EXERCICE 7 JOINTURE

Énoncé

On considère ces deux relations présentées dans un exercice précédent (voir figures 3.43 et 3.44).

Figure 3.43

Relation 'Film'.

Film(Prix, Format, Type, Nombre, Numero_Film)

Prix	Format	Type	Nombre	Numero_Film
12	4/3	Couleur	3	2
4	16/9	Noir/Blanc	1	4
12	16/9	Couleur	1 664	50
35	4/3	Noir/Blanc	890	12
12	16/9	Noir/Blanc	1	12

Énoncé

Catalogue(Numero_Film, Titre)

Figure 3.44

Relation 'Catalogue'.

Numero_Film	Titre
2	*Le train qui passe*
4	A toi !
56	Les chats du Sénégal
111	Le temps expliqué
12	*Les impôts faciles*

Trouvez la liste des titres de films et leur format. Voyez-vous une incohérence dans le résultat ? Pouvez-vous lors de cette opération détecter si la contrainte d'intégrité référentielle suggérée à l'exercice précédent a été respectée ? Est-il possible de faire une jointure entre ces deux relations sur le champ 'Prix' de la relation 'Film' avec le champ 'Numero_film' de la relation 'Catalogue' ? Si oui, que signifie le résultat ?

Solution

On fait une jointure entre les deux relations sur les champs Numero_Film (voir figure 3.45).

Figure 3.45

Jointure des relations 'Film' et 'Catalogue'.

Prix	Format	Type	Nombre	Numero _Film (Films)	Numero _Film (Catalogue)	Titre
12	4 :3	Couleur	3	2	2	*Le train qui passe*
4	16 :9	Noir/Blanc	1	4	4	A toi !
35	4 :3	Noir/Blanc	890	12	12	Les impôts faciles
12	16 :9	Noir/Blanc	1	12	12	Les impôts faciles

On projette sur les champs 'Titre' et 'Format' (voir figure 3.46).

Figure 3.46

Projection sur la jointure des relations 'Film' et 'Catalogue'.

Titre	Format
Le train qui passe	4 :3
A toi !	16 :9
Les impôts faciles	4 :3
Les impôts faciles	16 :9

Il n'est pas incohérent d'obtenir deux fois le même titre avec deux formats différents. En revanche, la contrainte d'intégrité référentielle n'est pas respectée, car on n'obtient pas la même cardinalité que la relation de départ lors de l'opération de jointure. On aurait dû obtenir une cardinalité de 5. On constate sur cet exemple que le contenu '50' du champ 'Numero_Film' ne se trouve pas dans la relation 'Catalogue'. Il manque donc un tuple dans le résultat.

La jointure sur les champs 'Prix' et 'Numéro_film' est possible, car ils sont de même type. Elle sera donc effectuée par le SGBD sans rejet. On obtient la relation suivante (voir figure 3.47).

Figure 3.47

Jointure « sans objet » des relations 'Film' et 'Catalogue'.

Prix	Format	Type	Nombre	Numero _Film (Films)	Numero _Film (Catalogue)	Titre
12	4 :3	Couleur	3	2	12	*Les impôts faciles*
4	16 :9	Noir/ Blanc	1	4	4	A toi !
12	16 :9	Couleur	1664	50	12	Les impôts faciles
12	16 :9	Noir/ Blanc	1	12	12	Les impôts faciles

Cette opération n'a aucune signification dans la réalité.

EXERCICE 8 AUTRE JOINTURE

Énoncé

Comment trouver la(les) valeur(s) qui ne respectent pas l'intégrité référentielle du champ Numéro_Film de la relation 'Film' par rapport à la relation 'Catalogue' dans l'exemple précédent ? Quels sont les films du catalogue qui ne sont pas utilisés dans la relation 'Film' ?

Solution

On fait une jointure externe cette fois entre les relations sur les champs Numero_Film. Attention, l'opération n'est pas symétrique, la jointure se fait à partir de la relation 'Film' sur la relation 'Catalogue' (voir figure 3.48)

Figure 3.48

Jointure externe des relations 'Film' et 'Catalogue'.

Prix	Format	Type	Nombre	Numero _Film (Films)	Numero _Film (Catalogue)	Titre
12	4 :3	Couleur	3	2	2	*Le train qui passe*
4	16 :9	Noir/ Blanc	1	4	4	A toi !
12	16 :9	Couleur	1664	50	NULL	NULL
35	4 :3	Noir/ Blanc	890	12	12	Les impôts faciles
12	16 :9	Noir/ Blanc	1	12	12	Les impôts faciles

Il suffit alors de faire une sélection sur le contenu d'un champ provenant de la relation 'Catalogue' dont le contenu est 'NULL', par exemple le champ 'Titre'. On projette ensuite sur le champ 'Numero_Film' provenant de la relation 'Film'. On obtient une relation à une ligne et une colonne (ou une relation de degré 1 et de cardinalité 1) [voir figure 3.49].

Figure 3.49

Sélection et projection de la jointure externe des relations 'Film' et 'Catalogue'.

Numero _Film
50

Pour détecter les films du catalogue non utilisés, on va cette fois faire une jointure externe symétrique de la précédente à partir de la relation 'Catalogue' sur la relation 'Film' (voir figure 3.49).

Figure 3.50

Jointure externe des relations 'Catalogue' et 'Film'.

Numero _Film (Catalogue)	Titre	Prix	Format	Type	Nombre	Numero _Film (Film)
2	Le train qui passe	12	4 :3	Couleur	3	2
4	A toi !	4	16 :9	Noir/ Blanc	1	4
56	Les chats du Sénégal	NULL	NUL	NUL	NUL	NUL
111	Le temps expliqué	NUL	NUL	NUL	NUL	NUL
12	Les impôts faciles	12	4 :3	Noir/ Blanc	890	12

De même que précédemment, il suffit de faire une sélection sur le contenu d'un champ provenant de la relation 'Film' dont le contenu est 'NULL', par exemple le champ 'Prix'. On projette ensuite sur le champ 'Numero_Film' provenant de la relation 'Film'. On obtient une relation à deux lignes et une colonne (ou une relation de degré 1 et de cardinalité 2) [voir figure 3.51].

Figure 3.51

Sélection et projection de la jointure externe des relations 'Catalogue' et 'Film'.

Titre
Les chats du Sénégal
Le temps expliqué

Cet exercice permet de mettre en évidence l'asymétrie de l'opération de jointure externe et son rôle indispensable pour répondre à des questions de ce type. C'est pour cette raison qu'elle a été introduite dans le langage SQL dans un second temps, même si elle n'en faisait pas partie à l'origine.

EXERCICE 9 CALCUL SUR DES AGRÉGATS

Énoncé

Quelle est la moyenne des prix pour les films par format ?

Solution

On va commencer par une opération qui consiste à constituer autant de sous-relations que l'on trouve de valeurs différentes dans le champ 'Format' de la relation 'Film '. Ici, il y a deux

valeurs pour toute la relation '4 :3' et '16 :9'. L'opération s'appelle « réaliser des agrégats par rapport au contenu d'un champ ». Ensuite, pour ces deux sous-relations, on calcule la moyenne des valeurs du champ 'Prix'. Dans la relation résultat, on a le champ projeté 'Format' et le champ calculé 'Moyenne' (voir figure 3.52).

Figure 3.52

Agrégat sur la relation 'Film'.

Prix	Format
29,5	4 :3
9,3333	16 :9

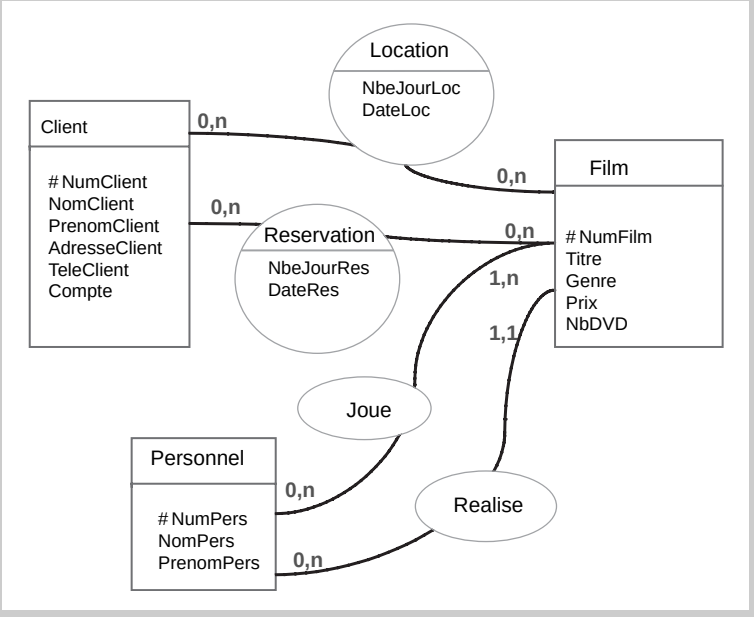
EXERCICE 10 PASSAGE DU MODÈLE ENTITÉ-ASSOCIATION AU RELATIONNEL

Énoncé

À partir du modèle entité-association modélisant une location de DVD, effectuez le passage au modèle relationnel (voir figure 3.53).

Figure 3.53

Modèle entité-association de la location de DVD.



Solution

On applique les règles classiques de passage vues précédemment dans ce chapitre. Une entité devient une relation composée des champs de l'entité, ayant comme clé celle de l'entité.

Client (NumClient, Nom, Prenom, Adresse, Tel, Compte) ;

Films(NumFilm, Titre, Genre, Prix, NombreDVD) ;

Personnels(NumPers, Nom, Prenom).

Les associations deviennent des relations ayant comme champs ceux de l'association et comme clé celles des entités associées.

Locations(DateLoc, NbeJourLoc, Livraison, NumFilm, NumClient) ;

Reservations(DateRes, NbJourRes, NumFilm, NumClient) ;

Joue(NumPers, NumFilm) ;

Realise(NumPers, NumFilm).

On remarque que l'association 'Realise' a une cardinalité de type '1-1' lors de son association avec Films. Cela signifie dans le monde réel qu'un film a un et un seul réalisateur.

On est dans le cas où l'association 'Realise' est « absorbée » par la relation créée à partir de l'entité associée, ici 'Film'. On obtient la relation suivante :

Films(NumFilm, Titre, Genre, Prix, NombreDVD, NumPers) et la relation 'Realise' disparaît.

On peut noter que les clés des relations créées à partir des associations 'Locations' et 'Reservations' supposent qu'un client ne réserve et ne loue pas le même film deux fois. Dans le cas contraire, on serait obligé d'ajouter un champ identifiant pour ces deux relations. Là encore, on ne peut décider cela qu'à l'isssue de discussions avec les utilisateurs de la base de données.

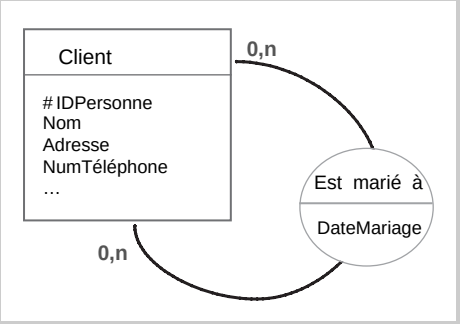
EXERCICE 11 PASSAGE DU MODÈLE ENTITÉ-ASSOCIATION AU RELATIONNEL II

Énoncé

À partir du modèle entité-association modélisant le lien de mariage, effectuez le passage au modèle relationnel (voir figure 3.54).

Figure 3.54

Modèle entité-association du mariage.



Solution

Le fait que l'association se fasse sur une même entité ne pose pas de problèmes particuliers ; on applique les règles classiques. On obtient deux relations :

- L'entité 'personne' devient une relation Personne(NumPersonne, Nom, Prenom, Adresse, Tel).
- L'association 'est_marié_à' devient une relation Mariage(NumPersonne_1, NumPersonne_2, DateMariage).

Les champs de l'association de même nom doivent être renommés car ils ne peuvent avoir le même nom.

Les cardinalités 0-n de l'association mariage ne signifient pas forcément que l'on accepte la polygamie, mais plutôt que l'on peut avoir été marié plusieurs fois à des dates différentes. À ce sujet, la clé de la relation 'Mariage' suppose qu'une personne ne se marie pas deux fois avec la même personne.

EXERCICE 12 NORMALISATION

Énoncé

Cette relation est-elle en première forme normale (voir figure 3.55) ?

Personne(Nom, Adresse_mail, Poste)

Figure 3.55

Relation 'Personne'.

Nom	Adresse_mail	Poste
André Dupont	adup@philips.com, andre@dupond.fr	Directeur
Stanislas De la Motte	sdlm@versailles.mairie.fr	Employé, assermenté
Elisabeth Macroix	elsa@yago.to, mac@rien.fr	Assistante

Solution

La relation n'est pas en première forme normale en raison des valeurs qui correspondent bien à des valeurs multiples contenues dans le champ 'Adresse_Mail'. En revanche, il n'est pas sûr que la valeur 'Employé, assermenté' de le champ 'Poste' soit une valeur multiple. Une virgule dans un champ n'induit pas nécessairement qu'il contient des valeurs multiples : c'est peut-être un contenu normalisé (un peu étrange certes).

On remarque que cette relation n'a pas non plus de clé candidate sérieuse, si l'on tient compte des dépendances fonctionnelles entre les champs. Or, seule la présence d'une clé permettra de faire la décomposition nécessaire à la normalisation. On ajoute un champ 'Ident_Personne' qui en sera la clé et l'on décompose en trois relations comme on l'a vu précédemment.

Une autre solution moins lourde est d'ajouter un champ supplémentaire pour l'adresse mail en supposant que l'on n'en stockera toujours que deux. On effectue alors la répartition des valeurs entre les champs. Il faut tout de même ajouter le champ 'Ident_Personne' pour que la relation possède une clé (voir figure 3.56).

Personne(Ident_Personne, Nom, Poste, Adresse_mail1, Adresse_mail2)

Figure 3.56

Relation 'Personne' modifiée.

Ident _Personne	Nom	Poste	Adresse_mail1	Adresse_mail2
1	André Dupont	Directeur	adup@philips.com	andre@dupond.fr
2	Stanislas De la Motte	Employé, assermenté	sdlm@ver-sailles.mairie.fr	sdlm@ver-sailles.mairie.fr
3	Elisabeth Macroix	Assistante	elsa@yago.to	mac@rien.fr

EXERCICE 13 NORMALISATION II

Énoncé

On considère la relation 'Film' précédente à laquelle on a ajouté le champ 'Support' (voir figure 3.57).

Film(Prix, Format, Couleur, Nombre, Numero_Film, Support)

Figure 3.57

Relation 'Film' avec support.

Prix	Format	Type	Nombre	Numéro_Film	Support
12	4 :3	Couleur	3	2	VHS
4	16 :9	Noir/Blanc	1	4	DVD
12	16 :9	Couleur	1664	56	DVD
35	4 :3	Noir/Blanc	890	12	VHS
12	16 :9	Noir/Blanc	1	12	DVD
99	Inconnu	Noir/Blanc	3	111	VHS

Après discussion avec les utilisateurs de la base de données, il ne peut y avoir deux fois le même film avec le même format dans cette relation qui est un état des stocks récapitulatif. À ce sujet, les utilisateurs indiquent que les formats '16 :9' seront toujours sur support 'DVD' et les formats '4 :3' et 'Inconnu' en support 'VHS'.

La relation est-elle en deuxième forme normale ?

Solution

La relation est en première forme normale.

Il n'y a pas de clé candidate 'atomique' pour cette relation.

D'après l'énoncé, on détermine que le couple de champs 'Numero_Film' & 'Format' est une clé. Or, il y a une dépendance fonctionnelle entre le support et le format. On a donc une dépendance entre un champ non clé et une partie de la clé ; la relation n'est pas en deuxième forme normale.

On décompose la relation de la manière suivante (voir figure 3.58).

Film(Prix, Format, Type, Nombre, Numero_Film)

Figure 3.58

Décomposition de la relation 'Film' avec support.

Prix	Format	Type	Nombre	Numero_Film
12	4 :3	Couleur	3	2
4	16 :9	Noir/Blanc	1	4
12	16 :9	Couleur	1664	50
35	4 :3	Noir/Blanc	890	12
12	16 :9	Noir/Blanc	1	12

FormatSup(Format, Support)

Format	Support
4 :3	VHS
16 :9	DVD
Inconnu	VHS

Les relations sont en première et en deuxième forme normale. On effectue une jointure pour reconstituer l'information de départ.

EXERCICE 14 NORMALISATION III

Énoncé

On considère la relation suivante que l'on a déjà étudiée au début du chapitre (voir figure 3.59).

Lecteur(Numero_carte, Nom, Age, Ville, Etablissement)

Figure 3.59
Relation 'Lecteur '.

Numéro_carte	Nom	Age	Ville	Etablissement
1	Henri	10	Paris	Université Sorbonne
2	Stanislas	34	Paris	Université Jussieu
3	Henriette	44	Lyon	CHU Bron
4	Dominique	19	Nancy	Université Poincaré
5	Isabelle	56	Nancy	INPL
6	Olivier	51	Marseille	Université Saint Charles
7	Henri	98	Paris	Université Sorbonne
8	Jerome	23	Nancy	INPL
9	Laurence	34	Bordeaux	Université Victor Segalen
10	Christian	41	Paris	Ecole Normale Supérieure
11	Antoine	16	Marseille	Université Saint Charles
12	Laurence	34	Paris	Université Jussieu

Est-elle en troisième forme normale ?

Solution

La relation est en première forme normale, il n'y a pas de champs multivalués.

La clé de cette relation est atomique. C'est le champ 'Numero_carte' qui a sans doute été ajouté à cet effet, sinon il n'y avait pas vraiment de clés candidates. Si la clé est atomique, alors on est sûr que la relation est en deuxième forme normale : il ne peut y avoir de dépendance entre un champ et une partie de la clé.

Il existe une dépendance fonctionnelle entre deux champs non clé 'Ville' et 'Etablissement' (voir discussion au début du chapitre). On est dans le cas d'une dépendance de type « transitif ». Le champ 'Ville' dépend du champ 'Etablissement' qui dépend de la clé 'Numero_carte'.

On décompose la relation de la manière suivante (voir figures 3.60 et 3.61).

Figure 3.60 **Lecteur(Numero_carte, Nom, Age, Etablissement)**

Relation 'Lecteur' décomposée I.

Numero_carte	Nom	Age	Etablissement
1	Henri	10	Université Sorbonne
2	Stanislas	34	Université Jussieu
3	Henriette	44	CHU Bron
4	Dominique	19	Université Poincaré
5	Isabelle	56	INPL
6	Olivier	51	Université Saint Charles
7	Henri	98	Université Sorbonne
8	Jerome	23	INPL
9	Laurence	34	Université Victor Segalen
10	Christian	41	Ecole Normale Supérieure
11	Antoine	16	Université Saint Charles
12	Laurence	34	Université Jussieu

Attention, la décomposition suppose que le nom de l'établissement est unique quelle que soit la ville. En effet, le champ 'Etablissement' est la clé de la deuxième relation. C'est ce qui a permis d'établir la relation de dépendance fonctionnelle. Cela ne peut être garanti sans discussion avec les utilisateurs de la base de données. Si ce n'était pas le cas, il faudrait utiliser un identifiant unique pour chaque établissement, qui se trouverait alors dans les deux relations et servirait à faire la jointure.

Figure 3.61 **Etablissement_Ville(Etablissement, Ville)**

Relation 'Lecteur' décomposée II.

Etablissement	Ville
Université Sorbonne	Paris
Université Jussieu	Paris
CHU Bron	Lyon
Université Poincaré	Nancy
INPL	Nancy
Université Saint Charles	Marseille
Université Victor Segalen	Bordeaux
École Normale Supérieure	Paris

Les deux relations sont en troisième forme normale. On effectue une jointure pour reconstituer l'information de départ.

Chapitre 4

SQL

Le langage SQL est l'un des éléments qui ont contribué au développement et au succès de l'approche relationnelle dans le monde des bases de données. En effet, la normalisation internationale du langage garantit la pérennité et la stabilité des données ainsi que des développements qui leur sont associés, indépendamment du SGBD et du langage utilisés.

Ce chapitre aborde les concepts et la syntaxe du langage SQL, et présente les trois grandes familles d'opérations que le langage permet d'exprimer :

- L'**interrogation** et la recherche dans les tables.
- La **gestion** de tables et de vues munies des contraintes associées. Ces instructions concernent la table et sa structure et constituent la partie LDD (*Langage de Description des Données*) de SQL.
- La **manipulation** de données. Ces instructions concernent les données contenues dans la table et constituent la partie LMD (*Langage de Manipulation des Données*) de SQL.

1 Concepts du langage SQL

Le langage SQL est essentiellement un « sous-produit » issu des travaux du groupe de travail System-R. Il s'agissait de la réalisation pratique des concepts de l'approche relationnelle chez IBM™. C'est une évolution du langage SEQUEL, lui-même dérivé du langage de recherche SQUARE.

Le langage est normalisé par l'ISO depuis 1987. La deuxième version du langage, SQL2, a été adoptée en 1992. Les exemples de cet ouvrage sont basés, dans la mesure du possible, sur cette norme. La troisième version du langage, SQL3, normalisée en 1999, ajoute essentiellement les fonctionnalités liées à l'utilisation de l'approche objet.

La quasi-totalité des SGBD disposent d'une interface SQL même si aucun ne couvre l'ensemble de la norme. La norme SQL prévoit trois niveaux de conformité : le niveau d'entrée, le niveau intermédiaire et le niveau complet. Les SGBD respectent en général le premier niveau et adoptent certains éléments des autres niveaux. La normalisation du langage garantit la portabilité générale des applications d'un SGBD à un autre, même s'il subsiste des différences entre les approches des différents éditeurs. En effet, compte tenu du temps nécessaire au processus de modification de la norme, certains éditeurs la devancent parfois et intègrent des fonctionnalités qui leur semblent essentielles… ou susceptibles de leur donner un avantage concurrentiel.

Le langage SQL manipule l'objet fondamental de l'approche relationnelle : la relation représentée par une table vue au chapitre 3. SQL est un langage dit « non procédural » ou « déclaratif », c'est-à-dire que l'on ne décrit pas la manière d'effectuer les opérations pas à pas : c'est le SGBD qui choisit la méthode utilisée pour y parvenir. C'est ce qui explique que des concours de rapidité de résolution de requête soient organisés chaque année pour tester les différentes stratégies des éditeurs. Par conséquent, il ne dispose pas d'instructions de structuration, telles que des boucles. Pour effectuer ce type d'opérations, on utilise un langage de programmation « classique », comme les langages C, php, Java™ et bien d'autres. Les instructions SQL sont alors intégrées dans le langage *via* une interface spécifique. Les résultats de la requête SQL sont alors stockés dans des structures de données propres au langage employé (par exemple un tableau) afin de pouvoir les manipuler. C'est typiquement ce procédé qui est employé pour construire une interface d'accès à une base de données par le Web.

Le langage de programmation qui intègre le langage SQL est alors appelé **langage hôte** (voir figure 4.1). De petites différences de syntaxe peuvent apparaître entre une requête SQL exprimée interactivement et la version « intégrée » dans un langage de programmation. Enfin, il existe une extension « procédurale » de SQL qui ajoute les fonctions, procédures et méthodes à SQL mais qui ne sera pas traitée dans le cadre de cet ouvrage.

Remarque

Un ensemble d'instructions SQL se nomme une requête. Une requête SQL se termine toujours par le caractère « ; ».

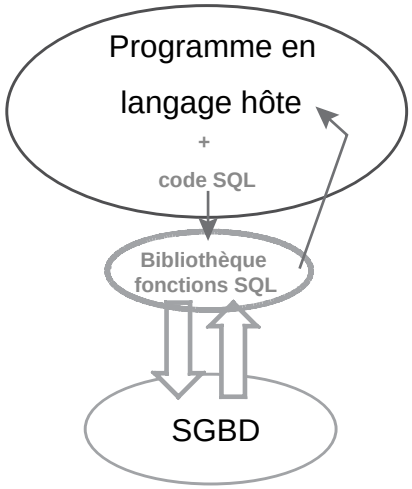


Figure 4.1

Interface entre SQL et les langages de programmation.

1.1 Organisation du chapitre

L'interrogation d'une ou de plusieurs tables est abordée dans la section « Opérations relationnelles avec SQL ». Toutes les requêtes de ce type retournent comme résultat une table qui peut être sauvegardée sous forme de table temporaire. Cette table intermédiaire est utilisée pour effectuer d'autres requêtes.

La création, la modification et la destruction d'une table sont abordées dans la section « Gestion de tables et de vues ». L'expression des contrôles de cohérence (contraintes d'intégrité) que l'on souhaite appliquer à une table est traitée logiquement dans cette section. La manipulation (insertion, modification et suppression) des données d'une table est abordée dans la section « Gestion des données ». La partie de SQL qui est consacrée à la gestion des autorisations sera abordée plus loin au chapitre 6.

Cet ouvrage ne traite pas de l'intégralité de la norme SQL ; ce n'est pas son objectif. Pour en savoir plus, vous pouvez vous reporter à l'ouvrage édité dans la même collection (F. Brouard et C. Soutou, *Syntex SQL*).

Remarque

Par convention, dans tous les exemples qui suivent, les instructions SQL sont indiquées en majuscules afin de les différencier des noms de tables et de champs. Les interpréteurs SQL ne différencient pas les majuscules des minuscules en ce qui concerne les instructions. En revanche, cela dépend du système d'exploitation utilisé ; le nom des champs ou des tables doit en général être écrit de manière exacte.

2 Opérations relationnelles avec SQL

Cette section présente l'expression en SQL des opérations de l'algèbre relationnelle vues précédemment : projection, sélection, agrégats, produits cartésiens et jointures. Une description précise de ces opérations se trouve au chapitre 3. Les exemples simples qui présentent les opérations relationnelles utilisent la base de données « casse » (voir figure 4.2).

Figure 4.2

Base de données ‘casse’.

Voiture

NumVoit	Marque	Type	Type
1	Peugeot	404	Rouge
2	Citroen	SM	Noire
3	Opel	GT	Blanche
4	Peugeot	403	Blanche
5	Renault	Alpine A310	Rose
6	Renault	Floride	Bleue

Personne

NumAch	Nom	Age	Ville	Sexe
1	Nestor	96	Paris	M
2	Irma	20	Lille	F
3	Henri	45	Paris	M
4	Josette	34	Lyon	F
5	Jacques	50	Bordeaux	M

Vente

DateVente	Prix	NumVoit	NumAch
1985-12-03	10 000	1	1
1996-03-30	70 000	2	4
1998-06-14	30 000	4	1
2000-04-02	45 000	5	2

2.1 PROJECTION (*SELECT*)

L’opération de projection consiste à sélectionner la (les) colonne(s) que l’on veut voir figurer dans la table « résultat ». On spécifie la liste des colonnes à inclure dans cette table derrière l’instruction SELECT en les séparant par des virgules. Si l’on désire afficher toutes les colonnes, on les désigne par le caractère « * ».

Projection sur les colonnes ‘Nom’ et ‘Ville’ de la table ‘Personne’.

```
SELECT Nom, Ville
FROM personne ;
```

Nom	Ville
Nestor	Paris
Irma	Lille
Henri	Paris
Josette	Lyon
Jacques	Bordeaux

Projection sur tous les champs de la table 'Personne'.

```
SELECT *
FROM personne ;
```

NumAch	Nom	Age	Ville	Sexe
1	Nestor	96	Paris	M
2	Irma	20	Lille	F
3	Henri	45	Paris	M
4	Josette	34	Lyon	F
5	Jacques	50	Bordeaux	M

Les colonnes de la table « résultat » peuvent être renommées par le mot clé AS.

```
SELECT Ville AS City
FROM personne ;
```

City
Paris
Lille
Paris
Lyon
Bordeaux

Valeurs distinctes d'une colonne

Une colonne « Salutations » ne devrait contenir que les valeurs normalisées « Madame », « Monsieur », « Mademoiselle ». L'affichage des valeurs distinctes permet de lister les différentes valeurs prises par la colonne pour repérer d'éventuelles incohérences comme la présence d'une valeur « Mr. », « M. », etc. Afin d'éliminer les doublons éventuels des valeurs d'une colonne de la table « résultat », on fait précéder le nom de la colonne par le mot clé DISTINCT.

Projection sur les valeurs distinctes du champ 'Marque' de la table 'voiture'.

```
SELECT DISTINCT Marque
FROM voiture ;
```

Marque
Peugeot
Citroen
Opel
Renault

Utilisation d'expressions pour créer une colonne

On rappelle qu'un grand principe en base de données est que l'on ne stocke pas dans une table ce qui peut être calculé. On évite ainsi l'occupation de place inutile ainsi que les problèmes d'incohérence provoqués par la mise à jour des données.

Par exemple, si l'on dispose d'une colonne 'prix' et d'une colonne 'quantité', on ne stocke pas la colonne 'chiffre d'affaire' on calcule ses valeurs à partir des colonnes précédentes (chiffre d'affaires = prix × quantité). Cette manière de procéder sera plus « coûteuse » en

calcul et pourra éventuellement affecter les performances du système. Les valeurs des colonnes (colonnes) de la table « résultat » peuvent être constituées par des expressions construites avec les opérateurs suivants (voir tableau 4.1).

Tableau 4.1

Opérateurs d'expressions de SQL.

+	**Addition**
-	Soustraction
*	Multiplication
/	Division
%	Modulo

Création d'une colonne 'Prix_Euros' dans la table 'vente' contenant le prix de vente en euros.

```
SELECT Prix, DateVente, (Prix / 6.5596) AS Prix_Euros
FROM vente ;
```

Prix	DateVente	Prix_Euros
10 000	1985-12-03	1 524.483 200
70 000	1996-03-30	10 671.382 401
30 000	1998-06-14	4 573.449 601
45 000	2000-04-02	6 860.174 401

SQL dispose de nombreuses autres fonctions intégrées, parfois dépendantes du SGBD utilisé, qui permettent par exemple le traitement des colonnes de types caractères, date…

Transformation d'une colonne 'Nom' de la table 'Personne' en majuscules.

```
SELECT UPPER(Nom) AS NomMajuscule
FROM personne ;
```

NomMajuscule
NESTOR
IRMA
HENRI
JOSETTE
JACQUES

Extraction du mois de la colonne 'DateVente' de la table 'Vente'.

```
SELECT MONTH(DateVente) AS Mois
FROM vente ;
```

Mois
12
3
6
4

Utilisation de fonctions statistiques sur toutes les valeurs d'une table

SQL ne possède pas d'instruction de « structuration » et ne permet donc pas de réaliser des boucles. Il possède des fonctions simples de traitement des données d'une colonne ; les calculs plus complexes seront réalisés à l'aide d'un langage de programmation comme on

l'a indiqué précédemment. Les colonnes (colonnes) de la table « résultat » peuvent être constituées de résultats de fonctions statistiques intégrées à SQL. Voici une liste (non exhaustive) des opérateurs statistiques de SQL (voir tableau 4.2) :

Tableau 4.2

Opérateurs statistiques de SQL.

COUNT	Comptage du nombre d'éléments (lignes) de la table
MAX	Maximum des éléments d'une colonne
MIN	Minimum des éléments d'une colonne
AVG	Moyenne des éléments d'une colonne
SUM	Somme des éléments d'une colonne

Remarque

Les fonctions statistiques s'appliquent à l'ensemble des données d'une colonne (sauf pour la fonction COUNT qui s'applique aux lignes de la table entière). Pour toutes ces opérations, la table « résultat » contiendra une seule ligne et souvent une seule colonne.

Calcul de la moyenne des prix de vente pour la table 'vente'.

```
SELECT AVG(Prix) AS Prix_Moyen
FROM vente ;
```

Prix_Moyen
38 750.000 0

Calcul du nombre de personnes (le nombre de lignes en réalité) de la table 'personne'.

```
SELECT COUNT(*) AS Nombre_Personne
FROM personne ;
```

Nombre_Personne
5

Dans le cas de la fonction COUNT, on ne spécifie pas la colonne sur laquelle s'applique la fonction puisqu'il s'agit de la table entière.

2.2 Sélection ou restriction (*WHERE*)

L'opération de sélection (ou restriction) consiste à indiquer un ou plusieurs critères pour choisir les lignes à inclure dans la table « résultat ». Ces critères utilisent évidemment le contenu des valeurs des colonnes. Le critère de sélection est indiqué à la suite du mot clé WHERE. Il est constitué d'expressions de conditions composées à l'aide d'opérateurs de comparaison et combinées à l'aide de connecteurs logiques.

Voici la liste des opérateurs de comparaison *classiques* permettant de constituer les expressions en SQL (voir tableau 4.3), la liste des opérateurs de comparaison spécifiques à SQL (voir tableau 4.4), et la liste des opérateurs et connecteurs logiques (voir tableau 4.5).

Tableau 4.3

Opérateurs de comparaison de SQL.

=	Égal
<>	Différent
<	Inférieur

>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

Extraction des enregistrements de la table vente dont le prix est supérieur à 50 000.

```
SELECT *
FROM vente
WHERE Prix > 50 000 ;
```

DateVente	Prix	NumVoit	NumAch
1996-03-30	70 000	2	4

Tableau 4.4

Opérateurs de comparaison spécifiques à SQL permettant de constituer des expressions.

BETWEEN <valeur> AND <valeur>	Appartient à un intervalle
IN <liste de valeurs>	Appartient à un ensemble de valeurs
IS NULL	Teste si la colonne n'est pas renseignée
LIKE	Compare des chaînes de caractères

Extraction des voitures blanches ou rouges.

```
SELECT *
FROM voiture
WHERE Couleur IN ("Blanc","Rouge") ;
```

NumVoit	Marque	Type	Couleur
1	Peugeot	404	Rouge

Extraction des personnes dont l'âge est compris en 40 et 60.

```
SELECT *
FROM personne
WHERE Age BETWEEN 40 AND 60;
```

NumAch	Nom	Age	Ville	Sexe
3	Henri	45	Paris	M
5	Jacques	50	Bordeaux	M

Tableau 4.5

Opérateurs et connecteurs logiques de SQL permettant de constituer des expressions.

AND	Et : les deux conditions sont vraies simultanément
OR	Ou : l'une des deux conditions est vraie
NOT	Inversion de la condition

Extraction des voitures de couleur « Blanche » ou de marque « Peugeot ».

```
SELECT *
FROM voiture
WHERE Couleur="Blanche" OR Marque="Peugeot" ;
```

NumVoit	Marque	Type	Couleur
1	Peugeot	404	Rouge
3	Opel	GT	Blanche
4	Peugeot	403	Blanche

Extraction des personnes n'habitant pas Paris.

```
SELECT *
FROM personne
WHERE NOT (Ville='Paris');
```

NumAch	Nom	Age	Ville	Sexe
2	Irma	20	Lille	F
4	Josette	34	Lyon	F
5	Jacques	50	Bordeaux	M

2.3 AGRÉGATS OU GROUPAGE (*GROUP BY*)

Les opérations d'« agrégation » ou de « groupage » regroupent les lignes d'une table par valeurs contenues dans une colonne. On applique généralement des opérations de type statistique sur les « sous-tables » ainsi créées. Pour réaliser cette opération avec SQL, on utilise le mot clé GROUP BY suivi du nom de la colonne sur laquelle s'effectue l'agrégat.

Affichage des différentes marques de voitures de la table 'voiture'.

```
SELECT Marque
FROM voiture
GROUP BY Marque ;
```

Marque
Citroen
Opel
Peugeot
Renault

On obtient dans ce cas le même résultat que si l'on avait utilisé le mot clé DISTINCT vu précédemment. L'utilisation courante de cette opération est d'appliquer en une seule instruction les fonctions statistiques déjà abordées aux différents sous-ensembles d'une table ainsi constitués.

Calcul du nombre de voitures des différentes marques de la table 'voiture'.

```
SELECT Marque, COUNT(*) AS Compte
FROM voiture
GROUP BY Marque ;
```

Marque	Compte
Citroen	1
Opel	1
Peugeot	2
Renault	2

Calcul de la moyenne d'âge par ville à partir de la table 'personne'.

```
SELECT Ville, AVG(Âge) AS Moyenne_Age
FROM personne
GROUP BY Ville ;
```

Ville	Moyenne_Age
Bordeaux	50.0000

Ville	Moyenne_Age
Lille	20.0000
Lyon	34.0000
Paris	70.5000

Restriction sur le résultat

Le résultat de l'opération de groupage peut lui-même être filtré : c'est-à-dire que l'on effectue une sélection des lignes par rapport au contenu des colonnes obtenues dans la table « résultat » précédente. En pratique, on filtre sur le résultat des opérations statistiques appliquées aux sous-ensembles définis par le groupage.

On reprend l'exemple précédent qui a permis de calculer le nombre de voitures par marques. On suppose que l'on élimine du résultat les marques dont on possède moins de deux voitures en considérant que ces marques ne sont pas représentatives du parc.

Calcul du nombre de voitures par marque de la table 'voiture' dont le nombre est supérieur à 1.

```
SELECT Marque, COUNT(*) AS Compte
FROM voiture
GROUP BY Marque
HAVING Compte > 1;
```

Marque	Compte
Peugeot	2
Renault	2

Remarque

Le mot clé HAVING permet d'effectuer une sélection sur le résultat de l'opération de groupage. Le mot clé WHERE opère une sélection sur les éléments (lignes) de la table avant l'opération de groupage..

Supposons que l'on veuille éliminer les voitures rouges de notre calcul.

Calcul du nombre de voitures par marque de la table 'voiture' dont la couleur est « Rouge ».

```
SELECT Marque, COUNT(*) AS Compte
FROM voiture
WHERE NOT (Couleur='Rouge')
GROUP BY Marque;
```

Marque	Compte
Citroen	1
Opel	1
Peugeot	1
Renault	2

2.4 REQUÊTES SUR PLUSIEURS TABLES

Lorsque l'on utilise plusieurs tables dans une requête SQL, il peut exister une ambiguïté dans les expressions sur les noms de colonnes. En effet, deux tables peuvent avoir une colonne de nom identique. Pour cette raison, on préfixera le nom de la colonne par le nom

de la table. Pour des questions de lisibilité, il est préférable de le faire systématiquement pour toutes les requêtes même si ce n'est pas absolument nécessaire.

Qualification des attributs par leur table d'appartenance.

```
SELECT voiture.Marque, voiture.Couleur
FROM voiture ;
```

Marque	Couleur
Peugeot	Rouge
Citroen	Noire
Opel	Blanche
Peugeot	Blanche
Renault	Rose
Renault	Bleue

Cette notation peut devenir rapidement fastidieuse si le nombre de tables est élevé et si leurs noms sont longs. Dans ce cas, on désigne la table par un alias plus commode, qui peut être réduit à une simple lettre, plutôt que par son nom complet. L'alias est indiqué simplement à la suite du nom de la table ou à l'aide du mot clé AS qui est optionnel.

Qualification simplifiée des attributs par leur table d'appartenance.

```
SELECT Vo.Marque, Vo.Couleur
FROM voiture AS Vo;
```

Marque	Couleur
Peugeot	Rouge
Citroen	Noire
Opel	Blanche
Peugeot	Blanche
Renault	Rose
Renault	Bleue

Produit cartésien

Le produit cartésien est la combinaison de toutes les lignes d'une table avec toutes les lignes d'une autre table sans tenir aucun compte du « sens » associé aux données. C'est une opération qui n'a guère d'intérêt en pratique. En SQL, cette opération s'écrit simplement.

Produit cartésien.

```
SELECT *
FROM personne, voiture ;
```

NumAch	Nom	Age	Ville	Sexe	Num Voit	Marque	Type	Couleur
1	Nestor	96	Paris	M	1	Peugeot	404	Rouge
2	Irma	20	Lille	F	1	Peugeot	404	Rouge
3	Henri	45	Paris	M	1	Peugeot	404	Rouge
4	Josette	34	Lyon	F	1	Peugeot	404	Rouge
5	Jacques	50	Bordeaux	M	1	Peugeot	404	Rouge
1	Nestor	96	Paris	M	2	Citroen	SM	Noire

NumAch	Nom	Age	Ville	Sexe	Num Voit	Marque	Type	Couleur
2	Irma	20	Lille	F	2	Citroen	SM	Noire
3	Henri	45	Paris	M	2	Citroen	SM	Noire
4	Josette	34	Lyon	F	2	Citroen	SM	Noire
5	Jacques	50	Bordeaux	M	2	Citroen	SM	Noire
1	Nestor	96	Paris	M	3	Opel	GT	Blanche
2	Irma	20	Lille	F	3	Opel	GT	Blanche
3	Henri	45	Paris	M	3	Opel	GT	Blanche
4	Josette	34	Lyon	F	3	Opel	GT	Blanche
5	Jacques	50	Bordeaux	M	3	Opel	GT	Blanche
1	Nestor	96	Paris	M	4	Peugeot	403	Blanche
2	Irma	20	Lille	F	4	Peugeot	403	Blanche
3	Henri	45	Paris	M	4	Peugeot	403	Blanche
4	Josette	34	Lyon	F	4	Peugeot	403	Blanche
5	Jacques	50	Bordeaux	M	4	Peugeot	403	Blanche
1	Nestor	96	Paris	M	5	Renault	AlpineA 310	Rose
2	Irma	20	Lille	F	5	Renault	AlpineA 310	Rose
3	Henri	45	Paris	M	5	Renault	AlpineA 310	Rose
4	Josette	34	Lyon	F	5	Renault	AlpineA 310	Rose
5	Jacques	50	Bordeaux	M	5	Renault	AlpineA 310	Rose
1	Nestor	96	Paris	M	6	Renault	Floride	Bleue
2	Irma	20	Lille	F	6	Renault	Floride	Bleue
3	Henri	45	Paris	M	6	Renault	Floride	Bleue
4	Josette	34	Lyon	F	6	Renault	Floride	Bleue
5	Jacques	50	Bordeaux	M	6	Renault	Floride	Bleue

Le nombre de lignes de la table « résultat » est égal au produit du nombre de lignes des deux tables. Les colonnes sont celles des deux tables simplement juxtaposées.

Jointure interne (*INNER JOIN*)

Il s'agit de l'opération de base de l'algèbre relationnelle. Elle permet de lier deux tables entre elles en introduisant un critère de « sens des données issu du monde réel » par opposition à l'opération précédente. Elle peut s'exprimer de plusieurs manières en SQL. La première est semblable à la restriction du produit cartésien précédent, mais dans ce cas la requête n'est généralement pas traitée de manière optimale par le SGBD.

Construction de la jointure des tables 'voiture' et 'vente' sur le critère d'égalité de la colonne 'NumVoit'.

```
SELECT voiture.Marque, voiture.Couleur, vente.Prix
FROM voiture, vente
WHERE voiture.NumVoit=vente.NumVoit ;
```

Marque	Couleur	Prix
Peugeot	Rouge	10 000
Citroen	Noire	70 000
Peugeot	Blanche	30 000
Renault	Rose	45 000

Une autre manière d'exprimer la jointure interne passe par un opérateur de jointure spécifique JOIN. Il faut bien sûr spécifier la colonne sur laquelle s'effectue la jointure.

```
SELECT voiture.Marque, voiture.Couleur, vente.Prix
FROM vente JOIN voiture ON voiture.NumVoit=vente.NumVoit ;
```

Le traitement de la requête est dans ce cas optimisé par le SGBD. C'est important, car l'opération de jointure est complexe à réaliser pour un SGBD et est coûteuse en temps et en ressources. Le nombre de lignes de la table « résultat » est égal cette fois au nombre de lignes contenues dans les deux tables pour lesquelles le critère d'égalité des colonnes est respecté. Il est bien sûr possible d'effectuer la jointure sur plus de deux tables : on indique alors les différents critères de jointure entre les tables.

Construction de la jointure des tables 'voiture', 'personne' et 'vente' sur les critères d'égalité des colonnes 'NumaAch' et 'NumVoit'.

```
SELECT vo.Marque, vo.Couleur, ve.Prix, pe.Nom, pe.Age
FROM voiture AS vo, vente AS ve, personne AS pe
WHERE (vo.NumVoit=ve.NumVoit) AND (pe.NumAch=ve. NumAch);
```

Marque	Couleur	Prix	Nom	Age
Peugeot	Rouge	10 000	Nestor	96
Citroen	Noire	70 000	Josette	34
Peugeot	Blanche	30 000	Nestor	96
Renault	Rose	45 000	Irma	20

ou

```
SELECT vo.Marque, vo.Couleur, ve.Prix, pe.Nom, pe.Age
FROM voiture AS vo JOIN vente AS ve JOIN personne AS pe ON (vo.NumVoit=ve.NumVoit)
AND (pe.NumAch=ve. NumAch);
```

Dans ce cas, il est indispensable de préciser le nom de la table pour chaque colonne dans la mesure où les tables ont des noms de colonnes en commun ; de plus, il peut être intéressant de les désigner par des alias pour une simple commodité d'écriture.

Remarque

L'opération de jointure précédente est dite également « interne » ; le mot clé INNER devant JOIN est, pour la plupart des SGBD, implicite et donc optionnel, mais il faudrait écrire en toute rigueur :

```
SELECT voiture.Marque, voiture.Couleur, vente.Prix
FROM vente INNER JOIN voiture ON voiture.NumVoit=vente.NumVoit ;
```

Jointure externe (*OUTER JOIN*)

L'opération de jointure interne ne permet pas de répondre à des questions du type : « Quelles sont les voitures qui n'ont pas été vendues ? » . À cette fin, il nous faut utiliser un opérateur capable d'inclure dans le résultat les lignes de la table 'voiture' qui n'ont pas de correspondance dans la table « vente » (par rapport aux valeurs de la colonne 'NumVoit') sans qu'il s'agisse d'un produit cartésien : cette opération spécifique se nomme la jointure externe.

L'opérateur SQL de jointure externe s'exprime par le mot clé OUTER JOIN. Cette opération n'est pas symétrique : soit on inclut toutes les lignes d'une table, soit toutes celles de l'autre. On précise cela à l'aide des mots clés LEFT et RIGHT ou en inversant simplement l'ordre des tables dans l'expression de l'instruction de jointure.

Dans la requête qui suit, toutes les lignes de la table 'voiture ' seront affichées, y compris celles pour lesquelles la colonne 'NumVoit' n'a pas de correspondance dans 'vente' : les colonnes issues de 'vente' ne pourront alors être mises en correspondance et auront la valeur NULL.

Construction de la jointure externe des tables 'voiture' et 'vente' sur les critères d'égalité de la colonne 'NumVoit'.

```
SELECT voiture.NumVoit, vente.NumVoit, voiture.Marque, voiture.Couleur, vente.Prix
FROM voiture LEFT OUTER JOIN vente ON voiture.NumVoit=vente.NumVoit ;
```

NumVoit	NumVoit	Marque	Couleur	Prix
1	1	Peugeot	Rouge	10 000
2	2	Citroen	Noire	70 000
3	NULL	Opel	Blanche	NULL
4	4	Peugeot	Blanche	30 000
5	5	Renault	Rose	45 000
6	NULL	Renault	Bleue	NULL

Si l'on opère la requête en inversant l'ordre des tables, ou en employant le mot clé RIGHT, on obtient la même réponse que pour la requête d'équi-jointure ci-dessus. Cela signifie qu'il n'y a pas de lignes dans 'vente' dont le contenu de la colonne 'NumVoit' ne possède pas de correspondance dans la colonne 'NumVoit' de la table 'voiture'. Ce résultat est prévisible puisqu'une voiture doit exister dans la table 'voiture' pour pouvoir faire l'objet d'une vente. On dispose alors d'un moyen de contrôler la cohérence des données entre les tables : dans notre cas, on pourra ainsi vérifier qu'il n'y a pas de valeur d'identifiant de voiture dans la table 'vente' (contenu de la colonne 'NumVoit' de la table 'vente') qui ne se trouve pas dans la table 'voiture' (contenu de la colonne 'NumVoit' de la table 'voiture').

Construction de la jointure externe inversée des tables 'voiture' et 'vente' sur les critères d'égalité de la colonne 'NumVoit'.

```
SELECT voiture.NumVoit, vente.NumVoit, voiture.Marque, voiture.Couleur, vente.Prix
FROM vente LEFT OUTER JOIN voiture ON voiture.NumVoit=vente.NumVoit ;
```

NumVoit	NumVoit	Marque	Couleur	Prix
1	1	Peugeot	Rouge	10 000
2	2	Citroen	Noire	70 000
4	4	Peugeot	Blanche	30 000
5	5	Renault	Rose	45 000

ou

```
SELECT voiture.NumVoit, vente.NumVoit, voiture.Marque, voiture.Couleur, vente.Prix
FROM voiture RIGTH OUTER JOIN vente ON voiture.NumVoit=vente.NumVoit ;
```

Il faut terminer la requête et répondre à la question de départ : « Quelles sont les voitures qui n'ont pas été vendues ? » Pour ce faire, il suffit de sélectionner les lignes dont l'une des colonnes issues de la table 'vente' n'a pas pu être mise en correspondance avec une ligne de la table 'voiture' : le contenu de cette colonne sera vide, ce qui signifie que l'on peut le tester avec le mot clé NULL. Par exemple, on teste le contenu de la colonne 'Prix' issue de la table 'vente'.

Sélection des lignes de la table précédente.

```
SELECT voiture.NumVoit, voiture.Marque, voiture.Couleur, vente.Prix
FROM voiture LEFT OUTER JOIN vente ON voiture.NumVoit=vente.NumVoit
WHERE vente.Prix IS NULL;
```

NumVoit	Marque	Couleur	Prix
3	Opel	Blanche	NULL
6	Renault	Bleue	NULL

2.5 Tri du résultat d'une requête

On utilise le mot clé ORDER BY pour spécifier la (les) colonne(s) sur laquelle (lesquelles) on souhaite trier le résultat.

Tri par Marque de la table 'voiture'.

```
SELECT Marque, Type
FROM voiture
ORDER BY Marque ;
```

Marque	Type
Citroen	SM
Opel	GT
Peugeot	404
Peugeot	403
Renault	Alpine A310
Renault	Floride

Il est possible de préciser l'ordre de tri par les mots clés ASC (croissant par défaut) ou DESC (décroissant).

Tri par Marque de la table 'voiture' en ordre décroissant.

```
SELECT Prix, DateVente
FROM vente
ORDER BY Prix DESC ;
```

Prix	DateVente
70 000	1996-03-30
45 000	2000-04-02
30 000	1998-06-14
10 000	1985-12-03

On peut indiquer plusieurs critères de tri, qui sont lus et traités de gauche à droite (ici, on trie d'abord par villes puis par âges).

Tri par Ville et par Age de la table 'personne'.

```
SELECT Nom, Age, Ville
FROM personne
ORDER BY Ville, Age ;
```

Nom	Age	Ville
Jacques	50	Bordeaux
Irma	20	Lille
Josette	34	Lyon
Henri	45	Paris
Nestor	96	Paris

3 Gestion de tables et de vues

3.1 TABLES

Le langage SQL comprend une partie manipulation de données (LMD) pour gérer les tables, qui est présentée dans cette section. Les opérations de création, de suppression et de modification des tables mettent à jour le **dictionnaire de données** du SGBD. On rappelle que le dictionnaire de données est une structure propre au SGBD qui contient la description des objets du SGBD (base de données, tables, colonnes, droits, etc.).

Remarque

Pour pouvoir gérer une table, il faut au préalable disposer des droits sur la base de données qui la contient : ces aspects sont abordés au chapitre 6.

Création

La création d'une table est une opération importante qu'il faut entreprendre avec soin. C'est lors de cette étape que l'on définit le type de données, la clé, les index éventuels et qu'il convient d'imposer des contraintes de validation garantissant la bonne qualité des informations entrées dans la table. La forme générale de l'instruction de création de table est la suivante :

```
CREATE TABLE <Nom de la table> ( liste des colonnes avec leur type séparé par ,) ;
```

Remarque

Le nom de la table ou d'une colonne ne doit pas dépasser 128 caractères. Il commence par une lettre, contient des chiffres, des lettres et le caractère « _ ». Attention de même à ne pas utiliser un mot clé SQL.

```
CREATE TABLE voiture (
NumVoit INT,
Marque CHAR(40),
Type CHAR(30),
Couleur CHAR(20)
) ;
```

Les tables peuvent être créées de manière temporaire : elles seront donc effacées à la fin de la session de l'utilisateur à l'aide du mot clé TEMPORARY.

Création de la table 'voiture'.

```
CREATE TEMPORARY TABLE temporaire (
Identifiant INT,
Jour DATE,
Valide BOOLEAN
) ;
```

Les tables peuvent être issues directement du résultat d'une requête en utilisant le mot clé AS : c'est particulièrement commode pour pouvoir disposer de résultats intermédiaires en fin de vérification, lors d'une série de manipulations sur une table.

```
CREATE TEMPORARY TABLE resultat
AS
(SELECT Vo.Marque, Vo.Couleur
FROM voiture AS Vo);
```

Type de données Le type de données est choisi essentiellement en fonction des opérations qui sont effectuées sur la colonne. Le choix du type permet également de mettre en place un premier niveau de restriction sur le contenu des données : une colonne de type numérique ne pourra pas contenir de caractères. Des restrictions plus fines seront définies à la section « Contraintes d'intégrité ».

Voici une liste (**non exhaustive**) des types de données SQL (voir tableaux 4.6, 4.7, 4.8 et 4.9).

Tableau 4.6

Types de données numériques de SQL.

INT	Entier standard (32 bits)
SMALLINT	Entier « petit » (16 bits)
REAL	Réel (taille spécifique au SGBD)
FLOAT(n)	Réel (représenté sur « n » bits)

Tableau 4.7

Types de données chaînes de caractères de SQL.

CHAR(n)	Chaîne de caractères de longueur « n » (codage ASCII 1 octet)
VARCHAR(n)	Chaîne de caractères de longueur maximale « n » (codage ASCII 1 octet)
NCHAR(b)	Chaîne de caractères de longueur « n » (codage Unicode sur 2 octets)
NVAR-CHAR(b)	Chaîne de caractères de longueur maximale « n » (codage Unicode sur 2 octets)

Tableau 4.8

Types de données date de SQL.

BOOLEAN	Booléen
BLOB	*Binary Large Object* : permet de stocker tout type binaire (photo, fichier traitement de texte…)

Tableau 4.9

Types de données binaires de SQL.

DATE	Date
TIME[(n)]	Heure, n (optionnel) est le nombre de décimales représentant la fraction de secondes

Suppression

La commande DROP TABLE permet de supprimer une table.

```
DROP TABLE voiture
```

Si la table est référencée dans une autre table (par exemple, contrainte d'intégrité référentielle), le SGBD refuse en général de la supprimer : il utilise l'option RESTRICT par défaut. Si l'on désire tout de même la supprimer ainsi que tous les objets qui lui sont liés, il faut alors utiliser l'option CASCADE. Dans l'exemple, si la table 'vente' utilise la table 'voiture' comme table de référence pour le contenu de la colonne 'NumVoit', on ne peut supprimer la table 'voiture' avant d'avoir supprimé la table 'vente'.

```
DROP TABLE voiture CASCADE
```

Modification

La commande **ALTER TABLE** permet de modifier la structure de la table, c'est-à-dire d'ajouter, de supprimer ou modifier des colonnes.

Ajout d'une colonne de nom 'enplus' de type 'INT' à la table 'voiture' (ADD COLUMN).

```
ALTER TABLE voiture
ADD COLUMN enplus INT ;
```

Affichage de la table 'voiture' modifiée.

```
SELECT * FROM voiture ;
```

NumVoit	Marque	Type	Couleur	enplus
1	Peugeot	404	Rouge	NULL
2	Citroen	SM	Noire	NULL
3	Opel	GT	Blanche	NULL
4	Peugeot	403	Blanche	NULL
5	Renault	Alpine A310	Rose	NULL
6	Renault	Floride	Bleue	NULL

Suppression de la colonne de nom 'Couleur' de la table 'voiture' (DROP COLUMN).

```
ALTER TABLE voiture
DROP COLUMN Couleur;
```

Affichage de la table 'voiture' modifiée.

```
SELECT * FROM voiture ;
```

NumVoit	Marque	Type	enplus
1	Peugeot	404	NULL
2	Citroen	SM	NULL
3	Opel	GT	NULL
4	Peugeot	403	NULL
5	Renault	Alpine A310	NULL
6	Renault	Floride	NULL

La commande ALTER permet de modifier également les contraintes associées aux colonnes. Cette partie est traitée à la section suivante, « Contraintes d'intégrité ». Le mot clé COLUMN est optionnel.

Remarque

Il n'est pas possible de modifier directement le nom d'une colonne ou son type. Il faut pour cela écrire une série d'opérations, en utilisant par exemple des colonnes temporaires.

Voici la suite d'instructions permettant la modification du nom de la colonne de nom 'Couleur' de la table 'voiture' en 'Teinte' en changeant son type. L'instruction UPDATE sera détaillée à la section « Gestion des données ».

Ajout de la colonne 'Teinte'.

```
ALTER TABLE voiture
ADD COLUMN Teinte CHAR(60);
```

Affichage de la table 'voiture' modifiée.

```
SELECT * FROM voiture ;
```

NumVoit	Marque	Type	Couleur	Teinte
1	Peugeot	404	Rouge	NULL
2	Citroen	SM	Noire	NULL
3	Opel	GT	Blanche	NULL
4	Peugeot	403	Blanche	NULL
5	Renault	Alpine A310	Rose	NULL
6	Renault	Floride	Bleue	NULL

Recopie des données de 'Couleur' dans 'Teinte'.

```
UPDATE voiture
SET Teinte=Couleur ;
```

Affichage de la table 'voiture' modifiée.

```
SELECT * FROM voiture ;
```

NumVoit	Marque	Type	Couleur	Teinte
1	Peugeot	404	Rouge	Rouge
2	Citroen	SM	Noire	Noire
3	Opel	GT	Blanche	Blanche
4	Peugeot	403	Blanche	Blanche
5	Renault	Alpine A310	Rose	Rose
6	Renault	Floride	Bleue	Bleue

Suppression de la colonne 'Couleur'.

```
ALTER TABLE voiture
DROP COLUMN Couleur;
```

Affichage de la table 'voiture' modifiée.

```
SELECT * FROM voiture ;
```

NumVoit	Marque	Type	Teinte
1	Peugeot	404	Rouge
2	Citroen	SM	Noire
3	Opel	GT	Blanche
4	Peugeot	403	Blanche
5	Renault	Alpine A310	Rose
6	Renault	Floride	Bleue

3.2 CONTRAINTES D'INTÉGRITÉ

Lors de l'étape de conceptualisation, on a défini la notion de « domaine », qui décrira l'ensemble des valeurs que peut prendre un attribut. Au niveau de SQL, une première approche du domaine est établie par le choix du type de la colonne, mais cela n'est pas assez restrictif en général.

SQL vous permet de définir des conditions de validité plus fines lors de la création de la table, que l'on nomme **contraintes d'intégrité**. C'est le SGBD qui applique ces conditions au moment de l'insertion, de la modification ou même de la suppression de données dans le cas ou ces dernières sont liées à d'autres tables. Cette étape est parfois fastidieuse, mais elle garantit la cohérence des données et évite de se retrouver avec des bases de données, conceptuellement correctes, mais inutilisables faute de données valides.

On peut distinguer différents types de contraintes sur les colonnes :

- les propriétés générales comme l'unicité ;
- les restrictions d'appartenance à un ensemble ;
- les dépendances entre plusieurs colonnes.

Propriétés générales

- La valeur de la colonne doit être renseignée absolument (NOT NULL).
- La valeur doit être unique comparée à toutes les valeurs de la colonne de la table (UNIQUE).

Lorsque les deux conditions précédentes sont réunies, la colonne peut servir à identifier un enregistrement et constitue donc une « clé candidate ». On rappelle qu'il ne peut y avoir qu'une seule clé que l'on désignera en SQL par le mot clé PRIMARY KEY. Ici, on indique que la colonne 'NumAch' est choisie comme clé de la table (donc implicitement unique et non nulle) et que la colonne 'Nom' doit toujours être renseignée.

```
CREATE TABLE personne (
NumAch INT PRIMARY KEY,
Nom CHAR(20) NOT NULL,
Age INT
) ;
```

Si aucune mention n'est précisée comme pour la colonne 'Age', elle peut être renseignée ou non. Attention, une colonne non renseignée (c'est-à-dire qui contient la valeur NULL pour SQL) signifie qu'elle ne contient aucune donnée, et non pas, par exemple, qu'elle

contient « 0 » pour une colonne de type « entier » ou un espace pour une colonne de type « caractère ».

Si la clé est constituée de plusieurs colonnes (elle est dite **composite**) ; on indique la liste des colonnes constitutives de la clé à la suite du mot clé PRIMARY KEY.

```
CREATE TABLE vente (
DateVente DATE,
PRIX INT,
NumAch INT,
NumVoit INT,
PRIMARY KEY (NumAch, NumVoit)
) ;
```

Condition d'appartenance à un ensemble

Il s'agit de décrire le **domaine** dans lequel la colonne pourra prendre ses valeurs. Un ensemble peut être décrit :

- En donnant la liste de tous ses éléments constitutifs (IN). L'ensemble des jours de la semaine ne peut être exprimé que de cette manière : « lundi », « mardi », etc. On vérifie que la colonne 'couleur' ne peut prendre que des valeurs « normalisées » : 'Rouge' 'Vert' ou 'Bleu'.

```
CREATE TABLE voiture(
NumVoit INT PRIMARY KEY,
Marque CHAR(30) NOT NULL,
Type CHAR(20),
Couleur CHAR(40),
CHECK Couleur in ('Rouge','Vert','Bleu')
```

- Par une expression (>, < , BETWEEN…). Par exemple, le prix doit être supérieur à 1 000. On vérifie que l'âge est compris entre 1 et 80.

```
CREATE TABLE personne
(NumAch INT PRIMARY KEY,
Nom CHAR(20) NOT NULL,
Ville CHAR(40),
AGE INT NOT NULL,
CHECK (Age BETWEEN 1 AND 80)
);
```

- Par une référence aux valeurs d'une colonne d'une autre table (REFERENCES). Les colonnes doivent être de même type et l'on ne peut plus détruire par défaut une table qui apparaît comme référence. On vérifie que les valeurs identifiantes des personnes 'NumAch' et des voitures 'NumVoit' de la table 'vente' existent bien dans les tables de référence 'personne' et 'voiture'.

```
CREATE TABLE vente (
DateAch DATE,
PRIX INT,
NumAch INT NOT NULL REFERENCES personne(NumAch),
NumVoit INT NOT NULL REFERENCES voiture(NumVoit),
PRIMARY KEY (NumAch, NumVoit)
) ;
```

Condition sur plusieurs colonnes (contrainte de table)

Lorsque l'on désire exprimer des contraintes plus élaborées impliquant plusieurs colonnes, on peut définir une contrainte de table en utilisant le mot clé CONSTRAINT. On vérifie que la colonne 'Age' et la colonne 'Ville' doivent être renseignées ou vides en même temps.

```
CREATE TABLE personne
(NumAch INT PRIMARY KEY,
Nom CHAR(20) NOT NULL,
Ville CHAR(40),
AGE INT,
CONSTRAINT la_contrainte CHECK ( (Age IS NOT NULL AND Ville IS NOT NULL) OR   (Age
IS  NULL AND Ville IS NULL) )
;
```

3.3 VUES (*CREATE VIEW*)

Une « vue » est le résultat d'une requête que l'on peut manipuler de la même façon qu'une table. On peut considérer une vue comme une table dynamique dont le contenu est recalculé à chaque utilisation. On utilise les vues pour des raisons de commodité – il n'est pas nécessaire que certains utilisateurs voient le modèle complet qui est parfois complexe – ou encore de sécurité/confidentialité en restreignant l'accès à certaines données. Dans cette optique, les vues viennent en complément de la gestion des droits d'accès, qui est traitée au chapitre 6.

```
CREATE VIEW personne_bis (NumAch, Nom, Age)
  AS
  SELECT NumAch, Nom, Age
  FROM personne ;
```

Les vues permettent également de mettre à jour les tables, à condition que les règles d'intégrité de(s) table(s) utilisées pour construire la vue soient respectées : en pratique, seules les vues concernant une seule table peuvent effectuer des mises à jour.

4 Gestion des données

Les commandes SQL qui sont présentées dans cette section concernent non plus la gestion de la structure des tables, mais celle des contenus. On dispose classiquement de trois opérations : l'**insertion**, la **suppression** et la **mise à jour,** pour gérer les données d'une table.

L'insertion se fait enregistrement par enregistrement, mais cela peut se révéler fastidieux. Cependant, l'insertion de données est possible également à partir de la lecture d'un fichier externe dont le format est accepté par le SGBD. Ces instructions dépendent donc du SGBD utilisé.

Les opérations de suppression et de modification des données se font à partir de critères de sélection des enregistrements (lignes) à modifier ou à supprimer. Par exemple, on peut décider de supprimer toutes les personnes qui habitent Paris. Autre exemple : on actualise le prix en euros de l'ensemble des ventes qui ont eu lieu après la date du passage à l'euro.

Ces critères s'expriment de la même manière que pour les opérations de sélection vues précédemment. Il est également possible d'utiliser le résultat d'une requête pour déterminer l'ensemble des valeurs d'une colonne afin d'effectuer cette sélection.

4.1 INSERTION (*INSERT INTO*)

L'insertion d'enregistrements dans une table peut se réaliser de plusieurs manières :

- enregistrement par enregistrement (INSERT INTO) ;
- en insérant la réponse à une requête SQL (INSERT INTO).

La commande pour insérer des données est de la forme générale suivante :

```
INSERT INTO <nom de la table> [ liste des colonnes ] VALUES <liste des valeurs>
```

Insertion d'un enregistrement dans la table 'voiture'

```
INSERT INTO voiture
(NumVoit, Marque, Couleur)
VALUES (10,'Triumph','Bleue') ;
```

Affichage de la table 'voiture' modifiée.

```
SELECT * FROM voiture ;
```

NumVoit	Marque	Type	Couleur
1	Peugeot	404	Rouge
2	Citroen	SM	Noire
3	Opel	GT	Blanche
4	Peugeot	403	Blanche
5	Renault	Alpine A310	Rose
6	Renault	Floride	Bleue
10	Triumph	NULL	Bleue

Si certaines colonnes sont omises, elles prendront la valeur 'NULL'.

Si la liste des colonnes est omise, on considère qu'il s'agit de la liste de celles prises dans l'ordre défini lors de la création de la table.

Insertion d'enregistrement(s) à partir du résultat d'une requête

La table dans laquelle on insère les données doit avoir le même nombre de colonnes que la table « résultat » de la requête (et le même type).

```
INSERT INTO voiture
SELECT NumVoit, Marque, Type, Couleur
FROM voiturebis
WHERE NumVoit>10;
```

Remarque

Pour être insérées, les valeurs des colonnes doivent respecter les contraintes d'intégrité associées à la table.

4.2 SUPPRESSION (*DELETE FROM*)

L'opération de suppression permet de supprimer un ensemble d'enregistrements (lignes) que l'on identifiera avec une expression identique aux conditions de sélection vues précédemment.

```
DELETE FROM voiture
WHERE Couleur='Rouge' ;
```

Affichage de la table 'voiture' modifiée.

```
SELECT * FROM voiture ;
```

NumVoit	Marque	Type	Couleur
2	Citroen	SM	Noire
3	Opel	GT	Blanche
4	Peugeot	403	Blanche
5	Renault	Alpine A310	Rose
6	Renault	Floride	Bleue

Attention, si l'on ne spécifie aucune condition, tous les enregistrements sont supprimés.

```
DELETE FROM personne ;
```

4.3 MODIFICATION (*UPDATE*)

Pour cette opération, il faut préciser :

- la (les) colonne(s) concernée(s) ;
- la (les) nouvelle(s) valeur(s) ;
- les enregistrements pour lesquels on modifiera ces valeurs.

De même que précédemment, on identifiera les enregistrements concernés par une expression de sélection. La valeur modifiée peut être statique ou calculée à partir des valeurs d'autres colonnes.

Modification du nom d'une ville dans la table 'personne'.

```
UPDATE personne
SET Ville='Paris-Centre'
WHERE Ville='Paris' ;
```

Affichage de la table 'personne' modifiée.

```
SELECT * FROM personne ;
```

NumAch	Nom	Age	Ville	Sexe
1	Nestor	96	Paris-Centre	M
2	Irma	20	Lille	F
3	Henri	45	Paris-Centre	M
4	Josette	34	Lyon	F
5	Jacques	50	Bordeaux	M

Résumé

Voici une synthèse des commandes SQL présentées dans ce chapitre. Le langage SQL permet :

- La **gestion** de tables et de vues munies des contraintes associées (LDD, *Langage de Description des Données*). Ces instructions concernent la table et sa structure.

```
création   CREATE TABLE/VIEW <nom de la table>
destruction DROP TABLE/VIEW <nom de la table>
modification  ALTER TABLE/VIEW <nom de la table>
```

- La **manipulation** de données (LMD, *Langage de Manipulation des Données*). Ces instructions concernent les données contenues dans les tables.

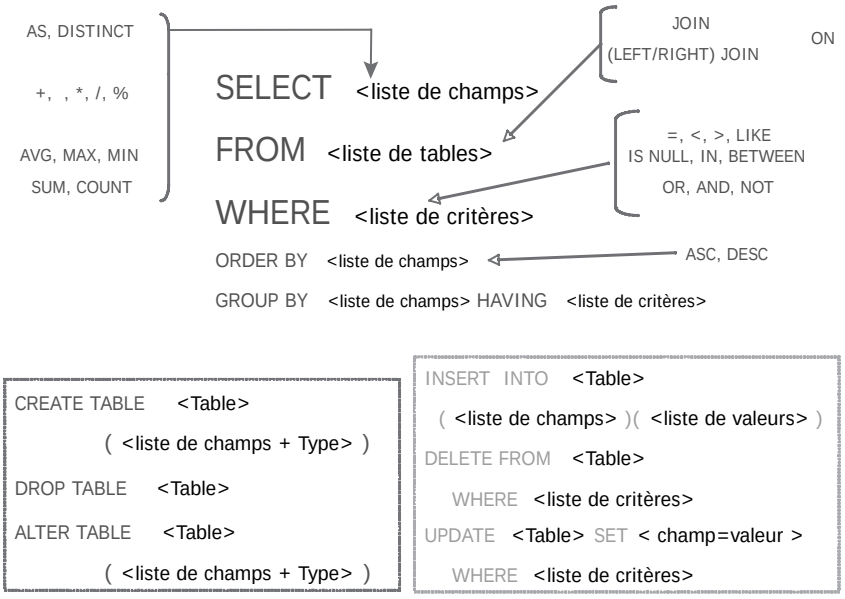
```
insertion INSERT INTO <nom de la table> (<liste de colonnes> <liste de valeurs>)
modification UPDATE <nom de la table> SET <colonne=valeur> WHERE <critère>
destruction DELETE FROM <nom de la table> WHERE <critère>
```

- L'**interrogation** et la recherche dans les tables.

```
SELECT <liste de colonnes>
FROM <nom de la table>
WHERE <critère>
```

Figure 4.3

Synthèse des commandes SQL.



Le langage SQL se révèle beaucoup plus complet que la partie qui est présentée dans ce chapitre. Le choix effectué parmi les commandes permet de répondre aux questions les plus courantes en base de données.

Exercices

EXERCICE 1 PROJECTION SIMPLE

Énoncé

Trouvez les différentes villes dans lesquelles habitent les personnes. Ordonnez le résultat par ordre décroissant.

Solution

Il s'agit d'une projection simple sur la colonne 'Ville' de la table 'personne' à l'aide du mot clé DISTINCT.

```
SELECT DISTINCT Ville
FROM personne
ORDER BY Ville DESC ;
```

EXERCICE 2 PROJECTION AVEC UNE COLONNE CALCULÉE

Énoncé

Affichez le chiffre d'affaires, c'est-à-dire la somme des prix de vente, toutes taxes (20 %) en considérant que la table contient les prix hors taxes. Renommez la colonne « Résultat » en « CA_TTC ».

Solution

Solution

Il s'agit d'utiliser à la fois une fonction statistique et une colonne calculée à partir de la colonne 'Prix' de la table 'vente'.

```
SELECT SUM(Prix*1.2) AS CA_TTC
FROM vente ;
```

EXERCICE 3 PROJECTION/RESTRICTION AVEC UN OPÉRATEUR STATISTIQUE

Énoncé

Trouvez l'âge moyen des personnes habitant Paris dans la base de données 'casse'.

Solution

Toutes les informations nécessaires se trouvent dans la table 'personne'.

Pour résoudre cet exercice, on peut raisonner en deux temps :

- Extraire de la table 'personne' les enregistrements dont la colonne 'Ville' contient « Paris' (sélection).

```
SELECT *
FROM Personne
WHERE Ville='Paris' ;
```

- Calculer à l'aide d'une fonction statistique la moyenne de la colonne 'Age' pour ses enregistrements (projection).

```
SELECT AVG(Age)
FROM Personne
WHERE Ville='Paris' ;
```

Pour compléter cet exercice, il serait plus élégant de renommer la colonne ainsi calculée, par exemple en « Moyenne_Paris » :

```
SELECT AVG(Age) AS Moyenne_Paris
FROM Personne
WHERE Ville='Paris' ;
```

EXERCICE 4 AGRÉGAT

Énoncé

Trouvez le nombre de voitures par marques dans la base de données 'casse'.

Solution

Toutes les informations nécessaires existent dans la table 'voiture'.

De même que pour l'exercice précédent, il est intéressant de raisonner en deux temps :

- Grouper les données par marques de voitures (agrégat).

```
SELECT Marque
FROM Voiture
GROUP BY Marque;
```

- Calculer à l'aide d'une fonction statistique le nombre de lignes par marque.

```
SELECT Marque, COUNT(*) AS Nombre
FROM Voiture
GROUP BY Marque;
```

EXERCICE 5 QUESTION NÉGATIVE

Énoncé

Trouvez l'âge des personnes qui n'habitent pas Paris.

Solution

Toutes les informations nécessaires sont présentes dans la table 'personne'.

Le problème est toujours d'exprimer une condition négative, ici la non-appartenance à un ensemble. Dans ce cas, c'est assez simple puisque toutes les valeurs de la colonnes 'Ville' sont renseignées. Ce serait plus difficile si certains enregistrements possédaient une colonne 'Ville' vide (valeur 'NULL') : il faudrait logiquement les exclure du résultat. Il est prudent d'inclure dans la réponse, en phase de vérification, la colonne sur laquelle on exprime un critère (ici, la colonne 'Ville'). Il est en effet assez facile de faire une erreur dans l'expression d'un critère.

```
SELECT Age, Ville
FROM personne
WHERE NOT (Ville='Paris') ;
```

EXERCICE 6 PRODUIT CARTÉSIEN

Énoncé

Écrivez l'expression du produit cartésien de la table 'voiture' et de la table 'personne'.

Combien de lignes et de colonnes possède la table « résultat »? Est-ce une opération « symétrique » (dans laquelle on retrouve le même résultat en inversant l'ordre des tables).

Solution

Il n'y a aucun critère de projection, ni de sélection ; toutes les colonnes et les lignes des deux tables participent donc au produit cartésien. On obtient 30 lignes (6 × 5) et 9 colonnes (5 + 4).

```
SELECT *
FROM personne, voiture;
```

L'opération est évidemment symétrique. En revanche, si les données sont les mêmes, elles ne se trouveront pas dans le même ordre.

EXERCICE 7 JOINTURE SIMPLE

Énoncé

Donnez les noms des personnes et la marque des voitures qu'elles ont achetées.

Solution

L'information se trouve dans les tables 'voiture' et 'personne'. Mais pour relier « sémantiquement » ces deux tables, on a besoin de la table 'vente'. Il faut donc faire une équi-jointure entre ces trois tables.

```
SELECT vo.Marque, pe.Nom
FROM voiture AS vo JOIN vente AS ve JOIN personne AS pe ON (vo.NumVoit=ve.NumVoit)
AND (pe.NumAch=ve. NumAch);
```

On aurait pu également écrire :

```
SELECT vo.Marque, pe.Nom
FROM voiture AS vo, vente AS ve, personne AS pe
WHERE (vo.NumVoit=ve.NumVoit) AND (pe.NumAch=ve. NumAch);
```

Le résultat est-il identique ? Pourquoi ?

EXERCICE 8 REQUÊTE SQL ÉTRANGE

Énoncé

Que donnerait cette requête ?

```
SELECT vo.Marque, pe.Nom
FROM voiture AS vo JOIN vente AS ve JOIN personne AS pe ON (vo.NumVoit=ve. NumAch)
AND (pe.NumAch=ve. NumVoit);
```

Obtient-elle une réponse ou provoque-t-elle un message d'erreur ?

Solution

C'est pratiquement la même requête que pour l'exercice précédent, mais le lien entre les tables a été réalisé sur des colonnes différentes. La syntaxe est correcte et le type des colonnes sur lesquelles a été faite la jointure est compatible ; SQL retournera donc une table « résultat ». Le fait que ce résultat n'a aucun sens du point de vue du monde réel ne peut être déduit du schéma des relations. Pour ce faire, il faut consulter le modèle conceptuel qui lui a servi de base.

EXERCICE 9 AUTRE QUESTION NÉGATIVE – JOINTURE EXTERNE

Énoncé

Quelles sont les villes où habitent les personnes qui n'ont pas acheté de voiture ?

Solution

C'est une question « négative », comme celle de l'exercice 4.5, mais qui met en jeu deux tables. Il s'agit de trouver, en comparant les valeurs de la colonne « NumVoit » comprises

dans les tables « vente » et « voiture », celles qui sont dans « voiture » et pas dans « vente ». À cet effet, on utilise une opération, qui n'est pas à proprement parler une opération de l'algèbre relationnelle, qui s'appelle la jointure externe qui permet d'afficher les enregistrements qui n'ont pas de correspondance dans l'autre table.

Là encore, le raisonnement peut se décomposer en deux étapes :

- Construction de la jointure externe des deux tables : l'ajout des colonnes 'NumAch' des deux tables dans la réponse sert simplement à vérifier notre requête.

```
SELECT personne.Ville, personne.NumAch, vente.NumAch
FROM personne LEFT OUTER JOIN vente ON personne.NumAch=vente.NumAch
;
```

- Sélection dans cette table « résultat » des éléments n'ayant pas de correspondance dans 'vente' et qui ont donc une valeur « NULL » pour la colonne 'NumAch'.

```
SELECT personne.Ville
FROM personne LEFT OUTER JOIN vente ON personne.NumAch=vente.NumAch
WHERE vente.NumAch IS NULL;
```

À noter que la question demandait d'établir la liste des villes. Si le contenu de la base de données était plus important, la même ville pourrait apparaître plusieurs fois dans la réponse. Pour éviter ce cas de figure, on précise que l'on veut la liste des différentes occurrences de 'ville' par le mot clé 'DISTINCT'.

```
SELECT DISTINCT personne.Ville
FROM personne LEFT OUTER JOIN vente ON personne.NumAch=vente.NumAch
WHERE vente.NumAch IS NULL;
```

EXERCICE 10 SÉLECTION SUR UN AGRÉGAT D'UNE JOINTURE

Énoncé

Calculez la moyenne des prix de vente par marques en ne considérant que les marques dont cette moyenne est supérieure à 40 000.

Solution

Les informations permettant de répondre à cette question se trouvent dans deux tables 'vente' et 'voiture'. Il nous faut donc effectuer une jointure sur la colonne 'NumVoit' pour lier ces deux tables.

```
SELECT vo.Marque, ve.Prix
FROM voiture AS vo JOIN vente AS ve ON vo.NumVoit=ve.NumVoit
;
```

Ensuite, on utilise l'opération d'agrégation pour effectuer un regroupement par 'Marque' et calculer pour chaque sous-ensemble la moyenne des prix de vente.

```
SELECT vo.Marque, AVG(ve.Prix)
FROM voiture AS vo JOIN vente AS ve ON vo.NumVoit=ve.NumVoit
GROUP BY vo.Marque
;
```

Enfin, on élimine du résultat du calcul précédent les marques dont la moyenne des prix est inférieure à 40 000 en ne gardant que les lignes dont la moyenne est supérieure ou égale à 40 000.

```
SELECT vo.Marque, AVG(ve.Prix) AS Moyenne
FROM voiture AS vo JOIN vente AS ve ON vo.NumVoit=ve.NumVoit
GROUP BY vo.Marque
HAVING Moyenne >= 40000
;
```

On peut utiliser l'alias du nom de la colonne calculée (ici « Moyenne ») pour effectuer la sélection. Attention, on ne peut pas utiliser le mot clé « WHERE » ici, car il s'agit d'une sélection sur le résultat du calcul et non pas *a priori* avant le calcul. On utiliserait « WHERE » pour répondre à une question du type : « Calculez la moyenne des prix de vente par marques en ne considérant pour ce calcul que les prix supérieurs à 40 000. »

```
SELECT vo.Marque, AVG(ve.Prix) AS Moyenne
FROM voiture AS vo JOIN vente AS ve ON vo.NumVoit=ve.NumVoit
WHERE ve.Prix >= 40000
GROUP BY vo.Marque;
```

Avec le jeu de données restreint dont on dispose, on remarque que le résultat est identique, mais c'est évidemment un hasard.

EXERCICE 11 SÉLECTION PAR RAPPORT AU RÉSULTAT D'UN CALCUL STATISTIQUE

Énoncé

Affichez les prix qui sont supérieurs à la moyenne.

Solution

Ici, on doit comparer les valeurs de la table 'vente' par rapport à une opération qui a été réalisée sur toutes les valeurs de la table. C'est une question simple en apparence, mais parfois difficile à traiter car tous les SGBD n'acceptent pas les requêtes imbriquées. Comme toujours, on peut décomposer le raisonnement en plusieurs étapes :

- Calculer la moyenne des prix.

```
SELECT AVG(vente.Prix) AS Moyenne_Prix
FROM vente
;
```

- Comparer les prix.

```
SELECT ve1.Prix
FROM vente AS ve1
WHERE ve1.Prix > (SELECT AVG(ve2.Prix) FROM vente AS ve2);
```

EXERCICE 12 CRÉATION D'UNE TABLE

Énoncé

Créez la table 'personne' avec les colonnes suivantes :

- NumAch. De type entier, clé de la relation.
- Nom. De type caractère, de taille 30, ne doit pas être vide.
- Age. De type entier, compris entre 16 et 100.
- Ville. De type caractère, de taille 40.
- Sexe. De type caractère, de taille 1, doit contenir « H » ou « F ».

Solution

```
CREATE TABLE personne
(NumAch INT PRIMARY KEY,
Nom CHAR(30) NOT NULL,
Age INT,
Ville CHAR(40),
Sexe CHAR(1),
CHECK (Age BETWEEN 16 AND 100),
CHECK Sexe IN ('H','F')
);
```

EXERCICE 13 INSERTION DE DONNÉES DANS UNE TABLE

Énoncé

Insérez les valeurs suivantes dans la table précédemment créée.

- NumAch : 100
- Nom : Essai
- Ville : Paris

L'enregistrement est-il validé ? (Satisfait-il aux contraintes d'intégrité ?)

Solution

On n'insère par toutes les valeurs des colonnes ; donc, il faut préciser la liste des colonnes pour lesquelles une valeur est présente dans l'ordre que l'on choisit.

```
INSERT INTO personne
(NumAch, Ville, Nom)
VALUES (100,'Paris','Essai') ;
```

L'enregistrement est valide ; aucune colonne ne contredit les contraintes définies précédemment : la clé 'NumAch' est unique, le nom est renseigné, les colonnes 'Age' et 'Sexe' ne sont pas renseignées, mais ne sont pas obligatoires.

EXERCICE 14 MODIFICATION DES DONNÉES D'UNE TABLE

Énoncé

Modifiez le prix de vente (+10 %) des voitures.

Solution

Les informations à modifier se trouvent simplement dans la table 'vente' bien que la question évoque les « voitures ». Il n'y a pas de critère dans ce cas puisque l'on modifie toute la table.

```
UPDATE vente
SET Prix=Prix*1.1 ;
```

EXERCICE 15 REQUÊTE COMBINÉE

Énoncé

Quelles sont les personnes qui n'ont pas acheté de voitures rouges ?

Solution

La question paraît simple, mais elle doit être bien analysée. On doit considérer deux catégories de personnes : celles qui ont acheté des voitures mais pas de voitures rouges ; celles qui n'ont acheté aucune voiture. Pour obtenir la liste des personnes qui n'ont acheté aucune voiture, on utilise une requête de type jointure externe. Les informations se trouvent dans les deux tables 'personne' et 'vente' .

```
SELECT personne.Nom
FROM personne LEFT OUTER JOIN vente ON personne.NumAch=vente.NumAch
WHERE vente.NumAch IS NULL;
```

Pour obtenir la liste des personnes qui ont acheté une voiture, mais pas rouge, on utilise une requête de type sélection sur une équijointure. Les informations se trouvent dans les deux tables 'personne' et 'voiture', que l'on doit relier par la table 'vente'.

```
SELECT pe.Nom
FROM voiture AS vo JOIN vente AS ve JOIN personne AS pe ON (vo.NumVoit=ve.NumVoit)
AND (pe.NumAch=ve. NumAch)
WHERE vo.Couleur != 'Rouge';
```

Chapitre 5

Du langage parlé à SQL

Ce chapitre présente un exemple complet de réalisation d'une base de données, étape par étape. On part de l'énoncé dans la vie réelle jusqu'à l'interrogation de la base à l'aide du langage SQL. Cet exemple permet de récapituler les concepts présentés dans les chapitres précédents, à savoir :

- l'analyse du monde réel et la création du modèle entité-association ;
- le passage au modèle relationnel ;
- la création des tables en SQL par l'intégration des contraintes de contenu ;
- l'interrogation de la base de données par des requêtes SQL.

On aborde également ici les différents aspects d'ordre pratique qui se posent à chaque étape de la réalisation d'une base de données.

1 Présentation de l'activité à modéliser

On veut modéliser la gestion d'une entreprise de fabrication et de livraison de pizzas à domicile : la société RaPizz. Il s'agit d'une société en franchise qui utilise des formats et des compositions de pizzas normalisés à partir d'un ensemble d'ingrédients déterminés. En d'autres termes, le client n'a pas la liberté de composer lui-même une pizza personnalisée ; il doit choisir dans le catalogue proposé.

Produits

Les produits vendus sont… des pizzas. Une pizza est caractérisée par son nom, les ingrédients qui la composent et son prix de base. Pour chaque pizza, il existe trois tailles : « naine », « humaine » et « ogresse ». La « naine » est 1/3 moins chère que le prix de base, c'est-à-dire la taille « humaine », et l'« ogresse » est 1/3 plus chère.

Mode de distribution

Les pizzas sont livrées par des livreurs qui circulent en voiture ou à moto et qui n'ont pas de véhicules attitrés. La base de données doit également permettre le suivi de l'activité des livreurs et des véhicules qu'ils utilisent.

Modalités de vente

Le mode de vente est du type prépayé : préalablement à toute commande, les clients doivent s'abonner au service et approvisionner leur compte. On vérifie le solde du compte avant de préparer et de livrer la commande.

Il existe deux systèmes de bonification :

- Une pizza gratuite est offerte au bout de 10 pizzas achetées.
- Toute pizza livrée en plus de trente minutes est gratuite.

Objectifs du système

Le but de cette base de données est de gérer l'activité quotidienne de vente et de livraison de pizzas :

- vérification du solde du compte et facturation aux clients ;
- suivi du chiffre d'affaires ;
- refus d'honorer les commandes pour lesquelles le solde du compte client est insuffisant ;
- non-facturation des pizzas gratuites (retard ou fidélité).

On veut également effectuer des statistiques diverses sur l'activité :

- identification du meilleur client ;
- identification du plus mauvais livreur (nombre de retards dans la livraison) et du véhicule utilisé ;
- identification de la pizza la plus ou la moins demandée ;
- identification de l'ingrédient favori…

Remarque

Afin de simplifier le problème, on considère que l'opération de base à modéliser dans cette activité est la vente d'une unique pizza. La notion de commande qui peut contenir plusieurs pizzas n'est pas prise en compte. On pourra faire évoluer le système plus tard si besoin est pour intégrer cet aspect. Il est en effet plus simple de raisonner en termes de ventes unitaires, que l'on peut agréger ensuite, plutôt que d'attaquer directement sur des commandes multiples.

2 Élaboration du modèle entité-association

Lors de cette étape, on détermine les éléments qui permettront de constituer les futures tables de la base de données. Pour élaborer ce modèle, on procède en deux temps :

- construction du graphe des entités reliées par les associations ;
- qualification des associations par leurs cardinalités.

Ce schéma est très important, car il représente en somme la documentation de la future base de données. L'ensemble de tables obtenues à la suite de l'opération de passage au schéma relationnel est généralement peu lisible et donne peu d'indications sur les liens entre les différents éléments. En effet, il est quasi impossible de déterminer quelles tables peuvent être jointes, du point de vue du sens des données dans la réalité, sans disposer du schéma entité-association associé.

2.1 Identification des entités et des associations

La première étape consiste à repérer les différentes entités que l'on doit considérer. Pour ce faire, il nous faut trouver les phrases simples qui identifient l'activité par rapport à l'énoncé. Parallèlement, on peut chercher quels objets concrets du monde réel semblent impliqués dans le système. Puis, pour chaque entité, on doit déterminer une clé parmi les attributs. Enfin, on caractérise les liens entre les entités par des associations.

Entités et attributs

On rappelle qu'il ne s'agit pas vraiment d'un processus scientifique basé sur des règles précises. Une certaine part d'intuition est nécessaire, d'autant plus qu'il n'existe pas une solution unique dans la majorité des cas.

On peut remarquer quelques mots clés dans la description générale : 'pizza', 'ingrédient', 'client', 'livreur', 'véhicule'. Ces descripteurs représentent des objets familiers du monde réel. En revanche, les mot clés 'taille', 'prix', 'compte', 'retard', 'nom' et autres sont clairement plus des qualifiants que des objets. Ce qui relie tous ces objets pour constituer la représentation de l'activité est la notion de commande. Cette dernière est l'un des seuls objets abstraits, comparée aux autres objets concrets du monde réel, comme peut l'être un véhicule par exemple.

À partir de ces constatations, voici une proposition de quelques phrases extraites ou déduites de la description générale qui permettent d'effectuer une première synthèse de l'activité :

- « Une pizza est constituée de plusieurs ingrédients. »
- « Un client passe une commande. »
- « Une commande est livrée par un et un seul livreur. »
- « Une commande est livrée par un et un seul véhicule. »

On en déduit l'existence des entités suivantes :

- commande ;
- client ;
- pizza ;
- livreur ;
- véhicule ;
- ingrédient.

On peut se poser la question de savoir si un ingrédient est une entité à part entière ou un simple attribut d'une pizza. L'entreprise considérée est une franchise ou tout est très normalisé, et l'on peut imaginer que la liste des ingrédients l'est aussi. Un même ingrédient entre certainement dans la composition de plusieurs pizzas. Il est donc préférable de séparer les ingrédients des pizzas ; l'association entre les deux entités est représentée par la phrase : « Une pizza est constituée d'ingrédients. » Notons à propos des ingrédients que le système ne doit gérer que les aspects commande et livraison ; l'on ne s'intéresse pas ici à la gestion des stocks d'ingrédients.

Il manque dans la description un nombre important de données qui vont constituer certaines entités, en particulier les entités 'véhicule' ou 'client'. Le cas est fréquent lors de la réalisation du processus ; il est alors nécessaire de poser de nouvelles questions pour combler ces lacunes. On imagine qu'à la suite d'un dialogue avec les différents acteurs de l'entreprise, on obtient les renseignements suivants :

- Un client est caractérisé par son nom et son adresse.
- Un livreur est caractérisé par son nom et son numéro de téléphone.
- Un véhicule est caractérisé par sa marque, son type et son numéro d'immatriculation.

Ces informations complémentaires permettent de déterminer quelques attributs des entités ainsi constituées. Les autres attributs des entités se trouvent dans l'énoncé : par exemple « une pizza est caractérisée par son nom et par son prix » ou « un ingrédient est caractérisé par son nom ». L'affectation d'un attribut à une entité n'est pas toujours évidente. On va créer un attribut 'compte' qui contiendra les informations de solvabilité du client puisque l'on fonctionne en mode prépayé. Il faut se poser la question : « est-ce une propriété caractéristique du client ? » La réponse dans notre cas est aisée et l'attribut sera affecté à l'entité 'client'.

Mais comment prendre en compte la taille de la pizza ? S'agit-t-il d'une propriété caractéristique de l'entité 'pizza' ou la taille est-elle associée à la commande… ou même au client ? Pour répondre à cette question, on doit considérer le moment ainsi que l'endroit où intervient la notion de taille et déterminer son utilité. La taille sert uniquement à pondérer le prix de base de la pizza : elle n'est donc pas associée à la pizza, dont le prix est fixe ; elle ne constitue pas non plus – on s'en doutait un peu – une caractéristique d'un client. On pourrait, à la limite, la considérer comme telle si l'on disposait d'une information du type : « les grandes tailles sont uniquement commandées par des hommes, les tailles normales par des femmes et les petites tailles par des enfants de sexe indifférencié ». Comme ce n'est pas le cas, on associe logiquement la taille à l'entité 'commande'.

Ce type de réflexion doit être engagé également pour modéliser les possibilités de bonification qui peuvent être obtenues soit en raison d'un retard de livraison soit par une capitalisation sous forme de points de fidélité. Par un raisonnement semblable, on détermine que le retard est associé à la commande alors que la fidélisation est associée au client. Si l'on récapitule, on obtient les entités munies de leurs attributs suivants :

- commande (DateCom, Taille, Retard) ;

- client (NomClient, Adresse, Compte, PointsRaPizz) ;
- pizza (NomPizza, Prix) ;
- livreur (NomLivreur, Téléphone) ;
- véhicule (NumImmat, Marque, Type) ;
- ingrédient (NomIngrédient).

Choix de la clé

Les attributs étant identifiés, on doit maintenant choisir une clé pour chaque entité. Si l'on prend l'entité 'client', il est clair qu'aucun attribut ne peut convenir pour constituer une clé. De même, l'association de plusieurs attributs 'NomClient-Adresse', 'NomClient-Compte', 'Compte-PointsRaPizz' et autres ne permet pas de créer une clé. On ajoute alors classiquement un attribut identifiant qui sert de clé : ce peut être un simple nombre ou un mélange de lettres et de nombres (alphanumériques) plus commode à mémoriser.

En utilisant ces mêmes arguments, on est amené à ajouter des attributs numériques comme clés pour les entités 'livreur' et 'commande'.

Pour l'entité 'pizza', on peut supposer que le nom de la pizza est significatif et qu'il peut donc servir de clé. En effet, le catalogue de pizzas proposé est restreint et codifié par le fait qu'il s'agisse d'une entreprise de type « franchisé ». Il n'y a pas d'intérêt pour le franchiseur, d'un simple point de vue commercial, à donner deux fois le même nom à une pizza. En pratique, même si le nom est unique, on pourrait décider de ne pas l'utiliser car le contenu est un peu long. L'entité 'véhicule' dispose d'un champ identifiant : son numéro d'immatriculation qui est par définition unique. Enfin, en ce qui concerne les ingrédients, il peut être moins évident que le nom seul de l'ingrédient suffise à l'identifier. Il est préférable de lui ajouter un numéro.

On obtient les entités suivantes munies de leurs attributs :

- commande (NumCommande, DateCom, Taille, Retard) ;
- client (NumClient, NomClient, Adresse, Compte, PointsRaPizz) ;
- pizza (NomPizza, Prix) ;
- livreur (CodeLivreur, NomLivreur, Téléphone) ;
- véhicule (NumImmat, Marque, Type) ;
- ingrédient (NumIngre, NomIngrédient).

Associations et attributs

On peut, à partir des phrases qui ont permis de repérer les entités, en déduire les associations suivantes :

- 'Livre' entre 'Livreur' et 'Commande ;
- Transporte' entre 'Véhicule' et 'Commande' ;
- 'Passe' entre 'Client' et 'Commande' ;
- 'Constitue' entre 'Pizza' et 'Commande ;
- 'Compose' entre 'Pizza' et 'Ingrédient'.

Il reste à déterminer les attributs éventuels des associations. Dans notre cas, le choix d'utiliser une entité 'commande' fait que l'on regroupe naturellement dans cette entité les attributs qui auraient pu se retrouver sur les associations.

Éventuellement, une date de commande pourrait être un attribut de l'association 'Passe' si elle était différente de la date de la commande. On peut cependant imaginer que ce genre

d'entreprise ne prend pas les commandes à l'avance : on commande une pizza lorsque l'on a faim. On se trouve donc dans le cas un peu particulier où aucune des associations ne possède d'attribut (voir figure 5.1).

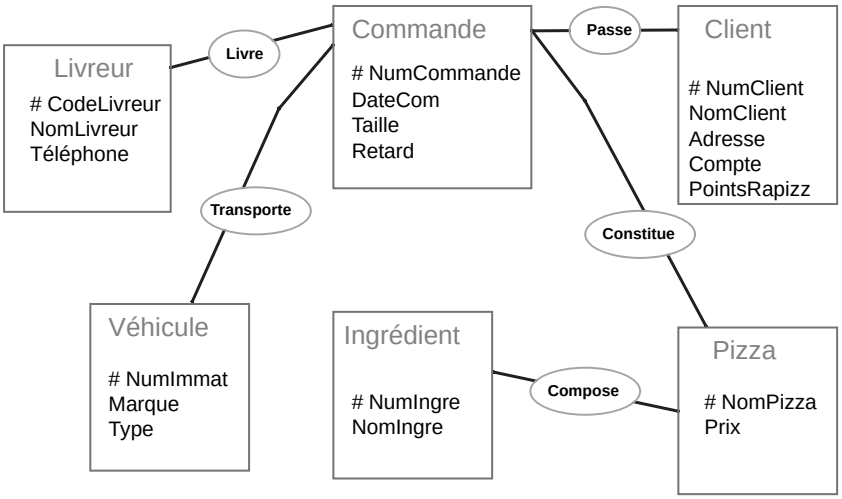


Figure 5.1

Modèle entité-association 'Livraisons de pizzas' sans cardinalités.

2.2 Détermination des cardinalités

On se pose ensuite des questions sur la nature des liens entre les entités capables de déterminer les cardinalités qui seront utilisées pour le passage au modèle relationnel. On doit trouver deux cardinalités, maximales et minimales, par entité associée, ce qui correspond à deux questions doubles par association. Il s'agit d'une étape d'inventaire un peu fastidieuse, mais elle est indispensable et permet de poser un autre regard sur le modèle.

Association 'Compose'

Ingrédient : un ingrédient peut-il ne jamais être utilisé dans la composition d'une pizza et peut-il être utilisé plusieurs fois ? On suppose que si un ingrédient est au catalogue, c'est qu'il est utilisé au moins une fois (1) et qu'il peut entrer dans la composition de plusieurs (n) pizzas. Les cardinalités associées sont de type '1-n'.

Pizza : une pizza peut-elle n'avoir aucun ingrédient et peut-elle en avoir plusieurs ? On suppose qu'une pizza est constituée d'au moins un (1) ingrédient et qu'elle peut en avoir plusieurs (n). Les cardinalités associées sont de type '1-n'.

Association 'Passe'

Client : un client peut-il n'avoir jamais passé de commandes et peut-il en avoir passé plusieurs ? Il peut y avoir une période pendant laquelle le client a approvisionné son compte mais n'a pas encore passé (0) de commande et il est évidemment encouragé à en passer plusieurs (n). Les cardinalités associées sont de type '0-n'.

Commande : une commande peut-elle avoir été passée par aucun client ou par plusieurs ? Une commande donnée est passée par un (1) et un (1) seul client. Les cardinalités associées sont de type '1-1'.

Association 'Livre'

Livreur : un livreur peut-il n'avoir jamais livré de pizzas et peut-il en avoir livré plusieurs ? Un livreur a au moins effectué une (1) livraison, sinon il n'est pas considéré comme tel. On peut imaginer que l'on ne rentre les informations associées à un livreur qu'à partir du moment où il a réellement effectué une livraison. Il est supposé faire plusieurs (n) livraisons. Les cardinalités associées sont de type '1-n'.

Commande : une commande peut-elle avoir été livrée par plusieurs livreurs ou par aucun ? Une commande est livrée par un (1) et un (1) seul livreur. Les cardinalités associées sont de type '1-1'.

Association 'Transporte'

Véhicule : un véhicule peut-il n'avoir jamais livré de pizzas et peut-il en avoir livré plusieurs ? Dans ce cas, la situation n'est pas tout à fait la même que pour les livreurs. On peut imaginer que les informations concernant un véhicule pourraient être insérées dans la base de données sans que le véhicule n'ait encore effectué une livraison (0). Si c'était le cas, on aurait des cardinalités de type '0-n'. On choisit ici des cardinalités associées de type '1-n'. Dans les deux cas, un véhicule est supposé être utilisé pour livrer plusieurs (n) commandes.

Commande : une commande peut-elle avoir été livrée par plusieurs véhicules ou par aucun ? Une commande est livrée par un (1) et un (1) seul véhicule. Les cardinalités associées sont de type '1-1'. On obtient le modèle entité-association suivant (voir figure 5.2) :

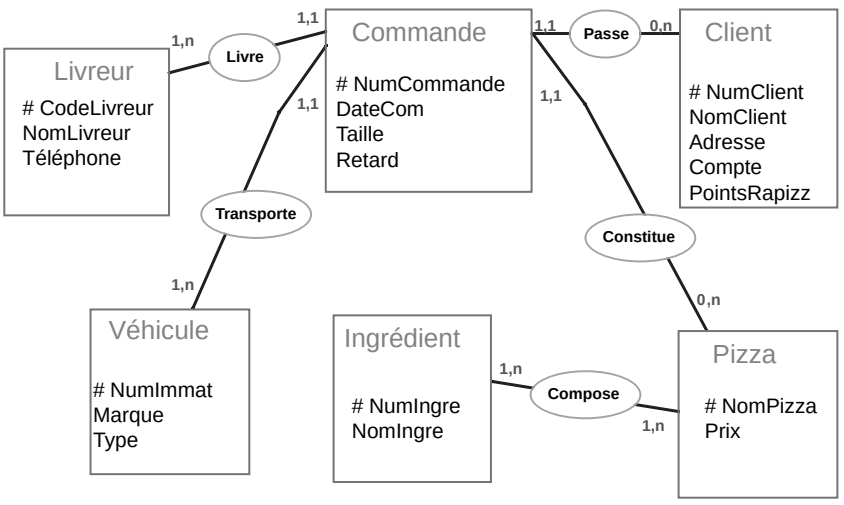


Figure 5.2

Modèle entité-association 'Livraisons de pizzas' avec cardinalités.

Remarque

Toutes les associations qui lient l'entité 'commande' sont de cardinalité '1-1'. On aurait pu ainsi considérer l'entité 'commande' comme une association qui serait alors quaternaire.

Cependant, si cela est possible, on préfère cependant éviter les associations autres que binaires, plus complexes à transformer en relations. De plus, le modèle UML ne propose pas de solution très cohérente pour représenter les associations ternaires ou de plus haut degré.

3 Passage au modèle relationnel

Cette section représente le travail préliminaire effectué en vue du passage à l'utilisation d'un SGBD. Elle comprend plusieurs parties :

- la transformation du modèle entité-association en modèle relationnel à l'aide des règles énoncées précédemment ;
- la vérification de la conformité des relations créées par rapport à la définition des formes normales du modèle relationnel ;
- la discussion sur le type des données à adopter pour chaque champ ainsi que sur les contraintes d'intégrités à définir ;
- la création des tables en SQL.

3.1 Transformation du modèle entité-association

Les deux règles générales de passage du modèle entité-association vers le modèle relationnel sont les suivantes :

- Une entité donne une relation de même clé que l'entité qui contient les mêmes attributs que l'entité.
- Une association donne une relation dont la clé est composée des deux clés des entités associées et des attributs de l'association.

Application de la règle générale

On obtient les relations suivantes à partir des entités :

- commande (NumCommande, DateCom, Taille, Retard) ;
- client (NumClient, NomClient, Adresse, Compte, PointsRaPizz) ;
- pizza (NomPizza, Prix) ;
- livreur (CodeLivreur, NomLivreur, Telephone) ;
- véhicule (NumImmat, Marque, Type) ;
- ingrédient (NumIngre, NomIngredient).

En appliquant la règle générale, on obtient les relations suivantes à partir des associations :

- livre (CodeLivreur, NumCommande) ;
- transporte (NumImmat, NumCommande) ;
- passe (NumClient, NumCommande) ;
- constitue (NomPizza, NumCommande) ;
- compose (NumIngre, NomPizza).

Cas particuliers des associations de cardinalité 1-1

Un cas particulier existe lorsque l'une des cardinalités d'une association est de type '1-1'. La relation représentant l'association disparaît et fusionne avec la relation représentant l'entité associée avec la cardinalité '1-1'. On peut dire qu'il y a « aspiration » de l'association par l'entité.

Dans cet exemple, de nombreuses associations sont de type « 1-1 » par rapport à l'entité « commande », comme l'association « livre ». L'application de cette règle produit une sim-

plification du nombre de relations au détriment de la lisibilité générale. On perd ainsi une partie de l'information sur le lien entre ces entités.

Ce processus peut être expliqué de la manière suivante. La clé de l'association 'livre' est composée des attributs 'CodeLivreur' & 'NumCommande'. Du fait de la cardinalité '1-1', on peut en déduire qu'à un numéro de commande correspond un et un seul code du livreur. La clé peut alors être simplifiée et réduite à l'attribut 'NumCommande'. On obtient la relation :

- livre (NumCommande, CodeLivreur)

Les deux relations 'commande' et 'livre' possèdent alors la même clé ; on peut les fusionner. On obtient la relation 'commande' augmentée suivante :

- commande (NumCommande, DateCom, Taille, Retard, CodeLivreur).

Les associations « transporte », « passe » et « constitue » sont de cardinalité « 1-1 » par rapport à l'entité « commande ». On peut aussi fusionner ces relations avec la relation « commande ». On obtient :

- commande (NumCommande, DateCom, Taille, Retard, CodeLivreur, NumImmat, NumClient, NomPizza).

Voici la liste des relations ainsi constituées :

- client (NumClient, NomClient, Adresse, Compte, PointsRaPizz) ;
- pizza (NomPizza, Prix) ;
- livreur (CodeLivreur, NomLivreur, Telephone) ;
- véhicule (NumImmat, Marque, Type) ;
- ingrédient (NumIngre, NomIngredient) ;
- commande (NumCommande, DateCom, Taille, Retard, CodeLivreur, NumImmat, NumClient, NomPizza) ;
- compose (NumIngre, NomPizza).

3.2 Vérification de la conformité aux formes normales

On doit vérifier que l'ensemble de relations créées à l'étape précédente est bien en conformité avec les formes normales. On peut se limiter en général aux trois premières formes normales. En effet, celle de Boyce-Codd provoque fréquemment un certain niveau de perte d'informations (voir chapitre 3 pour plus de précisions).

Première forme normale

A priori, aucune des relations ne dispose de champs à plusieurs valeurs (multivalués) : chaque champ possède une valeur « atomique ». Cependant, on pourrait s'interroger par exemple sur le contenu du champ 'NomClient' dans la relation 'client' ainsi que sur le champ 'NomLivreur' de la relation 'livreur'. S'ils contiennent le nom et le prénom de la personne, il est préférable de les séparer systématiquement en deux champs : 'NomClient' et 'PrenomClient', 'NomLivreur' et 'PrenomLivreur'. De même, il faut considérer le contenu du champ 'Adresse' de la relation 'client' : pour des raisons de simplification et de lisibilité de l'exemple, on n'a pas découpé l'adresse en ses différents composants classiques : 'numéro', 'type de voie', 'nom de la voie', 'ville', 'code postal' et 'pays'.

Toutes les relations sont en première forme normale si l'on tient compte des remarques sur le contenu de champs du précédent paragraphe.

Deuxième forme normale

La recherche de non-conformité à la deuxième forme normale n'a de sens que si la clé est composée de multiples champs. Dans notre cas, la seule relation qui possède une clé « composite » est la relation… 'compose'. De plus, cette relation ne contient aucun attribut qui ne fasse pas partie de la clé. La relation est en deuxième forme normale. Toutes les relations sont en deuxième forme normale.

Troisième forme normale

En revanche, la troisième forme normale concerne les liens qui peuvent exister entre des champs qui ne font pas partie de la clé d'une relation. Si l'on considère la relation 'client', les champs ne faisant pas partie de la clé sont les suivants : 'NomClient', 'Adresse', 'Compte', 'PointsRaPizz'. Il est clair que le solde du compte ne permet de déterminer ni le nom du client, ni son adresse, ni son nombre de points de fidélité. Ce dernier et le nom d'une personne ne permettent pas non plus d'identifier le contenu des autres champs. La question pourrait éventuellement se poser pour le champ 'Adresse', susceptible dans certains cas d'établir le nom du client si l'on suppose que deux clients n'habitent pas à la même adresse.

Par le même type de raisonnement sur la relation 'vehicule', on peut être amené à identifier une relation de dépendance entre le type qui est unique pour un véhicule. Si l'on suppose que les constructeurs déposent le nom du type de leur véhicule, il permet de déterminer sans ambiguïté le nom de la marque. Ce dernier pouvant être identique pour deux véhicules on évite ainsi la redondance provoquée par sa répétition. En toute rigueur, on serait conduit dans ce cas à décomposer la relation 'véhicule' en deux relations :

- véhicule (NumImmat, Type) ;
- constructeur (Type, Marque).

On remarque immédiatement la complexité introduite par cette décomposition en regard du gain apporté pour éviter une redondance modeste, surtout si le nombre de véhicules est relativement faible. Après analyse, on ne trouve de relations de dépendance entre aucun des champs de la relation 'commande' qui ne font pas partie de la clé, c'est-à-dire les champs 'DateCom', 'Taille', 'Retard', 'CodeLivreur', 'NumImmat', 'NumClient' et 'NomPizza'. S'il y avait de très mauvais livreurs qui livrent systématiquement en retard et d'autres qui ne livrent jamais en retard, il pourrait exister une relation de dépendance entre les champs 'CodeLivreur' et 'Retard', mais cela est peu probable dans la réalité.

On pourrait trouver une relation de dépendance entre les champs de cette relation en forçant un peu le raisonnement pour la relation 'Livreur'. En effet, si l'on suppose que le téléphone est un numéro de portable personnel et qu'il est associé à un livreur donné, alors le numéro de téléphone permet de déterminer le nom du livreur et son code client. Le champ devient une clé candidate. On pourrait peut-être supprimer la clé constituée par le code client pour éviter la redondance entre ce code que l'on a fabriqué et le numéro de téléphone. Il ne s'agit plus dans ce cas que d'un problème de conformité à la troisième forme normale puisque le champ clé est concerné. Dans la pratique, un critère pour la sélection entre plusieurs clés est de choisir le champ dont le contenu est le plus stable dans le temps. Un numéro de téléphone peut changer ; on lui préférera donc le numéro de code.

Toutes les relations ne sont pas strictement en troisième forme normale, mais on considère les redondances résiduelles comme acceptables.

Remarque

En pratique, on n'effectue pas toujours cette étape de manière stricte compte tenu de la complexité que produit la décomposition par le processus de normalisation. Il est parfois préférable de conserver un peu de redondance pour limiter le nombre de tables et préserver ainsi l'efficacité du système.

3.3 Types de données et contraintes

Cette section permet de déterminer quels contrôles doivent être appliqués sur le contenu des champs afin d'en garantir la cohérence. Les contraintes sont mises en œuvre par le SGBD employé et sont exprimées dans les instructions de création des tables. Ces contraintes expriment l'appartenance à un ensemble. Un ensemble en SQL peut être décrit de quatre manières :

- le typage général d'un point de vue des types prédéfinis de SQL (par exemple, entier, réel, caractère, date…) ;
- une énumération des différentes valeurs possibles (par exemple, « do », « ré », « mi », « fa », « sol »…) ;
- l'expression d'un intervalle dans lequel les valeurs sont contenues (par exemple, compris entre 10 et 30) ;
- la référence aux valeurs d'un champ d'une autre table (par exemple, contenu du champ 'code_postal' de la table 'communes').

Seule la description du type est obligatoire pour créer un champ, les autres contraintes étant évidemment optionnelles. Les champs dont les valeurs doivent être absolument renseignées sont indiqués spécifiquement en langage SQL par le mot clé « NOT NULL ».

Type(s) des données

La définition du type des données a pour but de faire une première restriction sur les valeurs que peut prendre un champ et surtout de spécifier les opérations et fonctions qu'il sera possible de lui appliquer. L'utilisation du type date pour le champ 'DateCom' permettra d'utiliser des fonctions d'extractions du mois, du jour de la semaine ainsi que d'autres opérations spécifiques aux données de type date pour ce champ. Le champ 'Telephone' de la table 'livreur' ne contiendra que des valeurs de type numérique, mais on n'effectuera jamais d'opérations de calculs sur ce champ : une moyenne des numéros de téléphone n'a guère de sens. En revanche, l'extraction d'un préfixe, comme par exemple les deux premiers chiffres d'un numéro peut être utile. On emploie alors un champ de type chaîne de caractères sur lequel il est possible d'employer des fonctions d'extraction de chaîne.

Les champs de type numérique sont presque tous de type entier, sauf le champ 'Compte' de la table 'client' et le champ 'Prix' de la table 'Pizza' qui sont de type réel. La taille des champs de type chaîne de caractères est un compromis à trouver entre l'occupation de place inutile et la taille maximale supposée des valeurs que peuvent contenir ce champ.

Table client

NumClient : code simple de type entier

NomClient : chaîne de caractères (à renseigner obligatoirement)

Adresse : chaîne de caractères

Compte : solde de type réel

PointsRaPizz : nombre de points de type entier

Table pizza

NomPizza : chaîne de caractères

Prix : solde de type réel (à renseigner obligatoirement)

Table livreur

CodeLivreur : code simple de type entier

NomLivreur : chaîne de caractères (à renseigner obligatoirement)

Telephone : chaîne de caractères (à renseigner obligatoirement)

Table vehicule

NumImmat : chaîne de caractères

Marque : chaîne de caractères (à renseigner obligatoirement)

Type : chaîne de caractères (à renseigner obligatoirement)

Table ingredient

NumIngre : code simple de type entier

NomIngrédient : chaîne de caractères (à renseigner obligatoirement)

Table commande

NumCommande : code simple de type entier

DateCom : type date (à renseigner obligatoirement)

Taille : chaîne de caractères (à renseigner obligatoirement)

Retard : chaîne de caractères (à renseigner obligatoirement)

CodeLivreur : code simple de type entier

NumImmat : chaîne de caractères

NumClient : chaîne de caractères

NomPizza : chaîne de caractères

Table compose

NumIngre : code simple de type entier

NomPizza : chaîne de caractères (à renseigner obligatoirement)

Contraintes d'intégrité

Les contraintes permettent de décrire de manière plus précise les ensembles auxquels appartiennent les champs.

Intervalle. Le champ 'Prix' de la table 'pizza' peut être limité à l'intervalle 1 .. 30. On pourrait définir un intervalle de validité pour les dates de commande (champ 'DateCom' de la table 'commande'), par exemple la date de commande doit être supérieure ou égale à la date du jour.

Énumération. Le champ 'Retard' de la table 'commande' doit contenir les valeurs 'O' ou 'N'. Le champ 'Taille' de la table 'commande' doit contenir les valeurs « naine », « humaine » ou « ogresse ». Comme il s'agit d'une franchise où les contenus sont norma-

lisés, on peut imaginer que l'ensemble des noms de pizzas est connu (par exemple, quatre saisons, margherita…). Le champ 'NomPizza' apparaît dans plusieurs tables, mais la contrainte de type énumération porterait uniquement sur le champ 'NomPizza' de la table 'pizza'. Les autres champs 'NomPizza' sont plutôt soumis à des contraintes de références comme on va le voir dans la partie consacrée aux références.

Référence au contenu d'une table. Les champs clés qui proviennent des associations transformées en relations font référence au contenu des relations qui proviennent des entités. Pour la relation 'compose', le champ 'NumIngre' fait référence au contenu du champ 'NumIngre' de la table 'ingrédient' et le champ 'NomPizza' au contenu du champ 'NomPizza' de la table 'pizza'.

En pratique, ici, la plupart des champs fusionnés de la relation 'commande' font aussi référence à des champs d'autres relations.

- 'CodeLivreur' de la relation 'commande'. Référence au champ 'CodeLivreur' de la relation 'livreur'.
- 'NumImmat' de la relation 'commande'. Référence au champ 'NumImmat' de la relation 'vehicule'.
- 'NumClient' de la relation 'commande'. Référence au champ 'NumClient' de la relation 'client'.
- 'NomPizza' de la relation 'commande'. Référence au champ 'NomPizza' de la relation 'pizza'.

3.4 Création des tables

La création des **tables** se fait à partir de la définition des **relations** effectuées précédemment. On spécifie le nom des champs, leur type, les contraintes d'intégrité et la définition des clés. On peut choisir également à ce moment les champs sur lesquels on souhaite créer des **index**. Ces derniers améliorent la recherche par le contenu, mais pénalisent les mises à jour puisqu'il faut modifier les tables d'index. En général, même si l'on est susceptible d'effectuer des recherches sur tous les champs, on ne crée pas d'index pour chacun d'entre eux afin de ne pas pénaliser les performances, même si l'on est susceptible d'effectuer des recherches sur tous les champs. Cela pourrait être cependant envisageable si le contenu de la base est très statique et que l'on souhaite favoriser les performances de recherche. Les index peuvent être créés séparément.

En pratique, on crée systématiquement un index sur les clés des relations, ne serait-ce que pour améliorer les opérations de jointure. Certains SGBD prennent d'ailleurs l'initiative de créer l'index sur la clé même si l'on ne le spécifie pas explicitement.

Table 'client'

```
CREATE TABLE client (
 NumClient INT PRIMARY KEY,
 NomClient VARCHAR(20) NOT NULL,
 Adresse VARCHAR(150) ,
 Compte REAL,
 PointsRaPizz INT
);
```

Table 'pizza'

```
CREATE TABLE pizza(
 NomPizza CHAR(30) PRIMARY KEY,
 Prix CHAR(30) NOT NULL,
```

```
 CHECK (Prix BETWEEN 1 AND 30)
);
```

Table 'ingredient'

```
CREATE TABLE ingredient(
 NumIngre INT PRIMARY KEY ,
 NomIngre CHAR(30) NOT NULL
);
```

Table 'vehicule'

```
CREATE TABLE vehicule(
 NumImmat CHAR(30) PRIMARY KEY,
 Marque CHAR(30) NOT NULL,
 Type CHAR(30) NOT NULL
);
```

Table 'livreur'

```
CREATE TABLE livreur (
 CodeLivreur INT PRIMARY KEY,
 NomLivreur CHAR(30) NOT NULL,
 TeleLivreur CHAR(30) NOT NULL
);
```

Table 'compose'

```
CREATE TABLE compose (
 NomPizza CHAR(30) REFERENCES pizza(NomPizza),
 NumIngre INT REFERENCES ingredient(NumIngre),
 PRIMARY KEY (NomPizza, NumIngre)
);
```

Table 'commande'

```
CREATE TABLE commande (
 NumCommande INT PRIMARY KEY,
 DateCom DATE,
 Taille CHAR(30) NOT NULL,
 Retard CHAR(1) NOT NULL,
 NumClient INT REFERENCES client(NumClient),
 NomPizza CHAR(30) REFERENCES pizza(NomPizza),
 CodeLivreur INT REFERENCES livreur(CodeLivreur),
 NumImmat CHAR(30) REFERENCES vehicule(NumImmat),
 CHECK Taille in ('Ogresse','Humaine','Naine'),
 CHECK Retard in ('O','N')
);
```

Exemple de création d'un index appelé 'Index_NumCommande' sur le champ 'NumCommande' de la table 'commande' dans l'ordre ascendant. L'instruction de création d'un index n'est pas toujours normalisée et la syntaxe peut différer suivant les SGBD.

```
CREATE INDEX Index_NumCommande ON commande (NumCommande ASC) ;
```

La création des tables ne peut pas se faire dans n'importe quel ordre. Il faut d'abord créer les tables auxquelles ont fait référence, puis seulement ensuite les autres tables.

En d'autres termes, on ne pourra créer la table 'commande' qu'après avoir créé les tables 'client', 'pizza', 'livreur' et 'vehicule'.

La destruction des tables se fait logiquement dans l'ordre inverse : on ne peut détruire une table qui est référencée par une autre table.

C'est le SGBD qui réalise ces vérifications.

4 Interrogation de la base de données

Cette section aborde la manière de passer d'une question en langage parlé à la requête SQL conduisant au résultat. L'ensemble des questions n'est évidemment pas exhaustif ; on a effectué un choix qui permet de mettre en valeur certains points jugés intéressants. On utilise également ces questions pour porter un regard sur les représentations choisies et vérifier qu'elles sont bien adaptées. Le sujet de l'optimisation des requêtes en fonction du SGBD choisi n'est pas traité, car il dépasse largement le cadre de cet ouvrage.

4.1 Menu

On veut extraire les données qui servent à imprimer la carte, ce qui signifie que l'on veut disposer du nom de chaque pizza, de son prix et des ingredients qui la composent. Pour ce faire, on commence par identifier les tables où se trouvent les champs que l'on veut projeter :

- Le nom de la pizza est dans la table 'pizza'.
- Le prix de la pizza est dans la table 'pizza'.
- Les noms des ingrédients se trouve dans la table 'ingredient'.

Le lien entre ces deux tables est matérialisé par le modèle entité-association : il s'agit de l'association 'compose'. Cette dernière est devenue la relation 'compose'. On effectue donc une double jointure entre les trois tables 'pizza', 'ingredient' et 'compose'. Les tables 'pizza' et 'compose' seront jointes sur le champ 'NomPizza' et les tables 'ingredient' et 'compose' seront jointes sur le champ 'NumIngre'.

Il y a deux manières d'écrire une jointure. Une jointure peut être La première consiste à l'appréhender comme une sélection sur un produit cartésien. L'écriture est plus pédagogique, mais totalement inefficace du point de vue du SGBD qui doit effectuer le produit cartésien puis sélectionner les lignes.

```
SELECT pizza.NomPizza, ingredient.NomIngre, pizza.Prix
FROM pizza, ingredient, compose
WHERE pizza.NomPizza = compose.NomPizza
AND ingredient.NumIngre = compose.Numingre;
```

La seconde, plus efficace, est d'utiliser l'instruction de jointure JOIN.

```
SELECT pizza.NomPizza, ingredient.NomIngre, pizza.Prix
 FROM pizza JOIN ingredient JOIN compose
 ON (pizza.NomPizza = compose.NomPizza
 AND ingredient.NumIngre = compose.Numingre);
```

On se trouve dans le cas très classique de deux entités liées par une association, qui donnent trois tables ('pizza' et 'ingredient' liées par 'compose'). Il sera nécessaire d'utiliser un langage de programmation pour obtenir un affichage élégant du nom de la pizza, de son prix et des ingrédients qui la composent. En effet, pour chaque nom de pizza, on obtiendra autant de lignes que d'ingrédients.

4.2 Fiche de livraison

On veut imprimer une fiche de livraison qui mentionne le nom du livreur, le type du véhicule utilisé, le nom du client, la date de la commande, le retard éventuel, le nom et le prix de base de la pizza. Par le même raisonnement que précédemment, on identifie les tables 'commande', 'client', 'pizza', 'livreur' et 'vehicule' pour les champs à projeter :

- ‘client’, pour le nom du client ;
- ‘pizza’, pour le nom et le prix de la pizza ;
- ‘livreur’, pour le nom du livreur ;
- ‘vehicule’, pour le type du véhicule ;
- ‘commande’, pour le retard, le numéro de commande et surtout pour établir le lien entre toutes les tables.

```
SELECT C.NumCommande, CI.NomClient, P.NomPizza, P.Prix, LI.NomLivreur, V.Type,
C.Retard
 FROM commande C JOIN livreur LI JOIN pizza P JOIN client CI JOIN vehicule V
 ON (C.CodeLivreur=LI.CodeLivreur AND P.NomPizza=C.NomPizza AND
 C.NumClient=CI.NumClient AND C.NumImmat=V.NumImmat)
 ORDER BY C.NumCommande
;
```

La table pivot est ici la table ‘commande’ qui assure la jointure avec toutes les autres tables. Cela confirme l’intuition que l’entité ‘commande’ aurait pu être considérée comme une association ‘n-aire’.

Sur une requête aussi simple, on remarque que le travail du SGBD est assez lourd puisqu’il doit effectuer la jointure de cinq tables. C’est la raison pour laquelle on préfère parfois utiliser des relations, et donc des tables, avec une certaine redondance afin d’améliorer les performances : l’opération de jointure est coûteuse. Une stratégie couramment employée consiste à gérer en interne une base de données sans redondance et à générer une table redondante qui servira à faire les requêtes. On dispose ainsi d’une garantie de cohérence et des performances préservées. Les bases de données en ligne accessibles par le Web sur lesquelles on effectue beaucoup de requêtes fonctionnent de cette manière. On réserve bien sûr cette méthode aux bases de données dont le contenu est assez stable : l’opération de regénération de bases à chaque changement est coûteuse.

4.3 Quels sont les véhicules n’ayant jamais servi ?

Si l’on se reporte aux cardinalités du modèle entité-association, le cas ne doit pas exister. On a choisi une cardinalité de type ‘1-n’, ce qui signifie qu’un véhicule a livré au moins une commande et qu’il a pu en livrer plusieurs. Cette requête va permettre de vérifier qu’il n’y a pas d’incohérence à ce niveau. On espère que le résultat sera vide.

On a besoin pour répondre aux questions du type « quels sont les… qui n’ont pas… » d’utiliser une extension des opérations de jointure : la jointure externe. Celle-ci permet d’inclure dans le résultat d’une jointure les enregistrements qui n’ont pas de valeur correspondante dans l’autre table. Les valeurs des champs de cette autre table sont alors logiquement vides. Il suffit de sélectionner ces lignes vides pour obtenir la réponse à la question. C’est une opération non symétrique ; on part de la table ‘vehicule’ que l’on va joindre.

```
SELECT vehicule.NumImmat, vehicule.Marque, vehicule.Type
 FROM vehicule LEFT OUTER JOIN commande
 ON vehicule.NumImmat = commande.NumImmat
 WHERE commande.NumImmat IS NULL;
```

Cette requête est typique des requêtes de vérification que l’administrateur d’une base doit effectuer pour procéder à certaines vérifications de cohérence. Cela est surtout vrai lorsque la base de données n’a pas été conçue en intégrant les contraintes d’intégrité indispensables.

4.4 QUELS CLIENTS COMMANDENT PLUS QUE LA MOYENNE ?

Bien que la question d'origine exprimée en langage parlé soit simple, ce genre de requête doit être abordée avec prudence. La meilleure approche consiste à séparer les étapes de réalisation de la requête ; cette dernière peut être dissociée en trois étapes :

- On calcule ce que commande chaque client.
- On calcule la moyenne des commandes.
- On cherche les clients qui ont commandé plus que la moyenne.

À chaque étape, on crée une table temporaire qui nous permet de réutiliser le résultat de la requête pour les autres étapes. Les tables temporaires disparaissent à la fin de la session SQL.

Calcul du nombre de commandes par client

On utilise la notion d'agrégat pour effectuer une opération statistique sur des regroupements d'enregistrements de la table. La présence du mot « par » dans la question oriente assez naturellement vers un regroupement et donc vers l'utilisation d'un agrégat.

```
CREATE TEMPORARY TABLE requete1
SELECT commande.NumClient, NomClient, COUNT(*) AS NombreCommande
FROM client JOIN commande ON client.NumClient=commande.NumClient
GROUP BY commande.NumClient;
```

Calcul de la moyenne des commandes

On doit calculer la moyenne du nombre de commandes par client ; on utilise à cet effet une opération statistique classique sur le champ de la table nombre de commande créée par la requête précédente.

```
CREATE TEMPORARY TABLE requete2
SELECT AVG(NombreCommande) AS MoyenneCommande
FROM requete1;
```

Extraction des clients ayant commandé plus que la moyenne

Enfin, on utilise la valeur de la moyenne obtenue par la seconde requête dans l'expression du critère de sélection des enregistrements sur la table obtenue par la première requête. Un petit artifice est employé ici. Comme la table provenant de la seconde requête ne contient qu'un champ, on effectue un produit cartésien entre ces deux tables sans modifier le nombre de lignes du résultat. Le but est de pouvoir disposer de la valeur du champ de la moyenne des commandes pour le comparer au compte du nombre de commandes.

```
SELECT requete1.NumClient, requete1.NomClient
FROM requete1, requete2
WHERE requete1.NombreCommande > requete2.MoyenneCommande;
```

On peut écrire ces différentes requêtes dans une seule en utilisant les **sous-requêtes** (ou requêtes imbriquées). Cependant, il est prudent d'accomplir d'abord les étapes précédentes. En décomposant, il est possible de vérifier les résultats à chaque étape et d'éviter ainsi un certain nombre d'erreurs.

```
SELECT A.NumClient, A.NomClient
FROM
(SELECT commande.NumClient, NomClient, COUNT(*) AS NombreCommande
 FROM client JOIN commande ON client.NumClient=commande.NumClient
 GROUP BY commande.NumClient) A
WHERE R1.NombreCommande >
       (SELECT AVG(B.NombreCommande)
```

```
            FROM (SELECT commande.NumClient, NomClient, COUNT(*) AS NombreCommande
               FROM client JOIN commande ON client.NumClient=commande.NumClient
                GROUP BY commande.NumClient) B
            )
;
```

Il s'agit exactement des mêmes requêtes que celles obtenues lors des étapes précédentes. Attention, tous les SGBD ne permettent pas d'imbriquer les requêtes.

4.5 Calcul du prix d'une commande

On désire calculer le prix d'une commande dans le but d'établir ensuite le récapitulatif du chiffre d'affaires mensuel, par exemple par client, et de façon générale d'autres calculs. Le prix d'une commande est fonction de la taille de la pizza commandée, qui pondère le prix de base de la pizza ; on peut l'extraire de la table 'pizza'.

Difficultés liées à la représentation choisie

On s'aperçoit lors de cette opération que la représentation sous forme textuelle (« naine », « humaine », « ogresse ») de la taille d'une pizza n'est pas adaptée pour le calcul. En effet, on est incapable de calculer son prix directement en SQL à partir des informations contenues dans la table.

Il n'y a pas d'indication dans la base de données du mode de calcul du prix de la pizza, ce qui signifie que l'on doit faire un test de ce type : si taille vaut « ogresse », alors prix = prix × (1 + 1/3). Le plus simple est alors de passer par un langage de programmation pour associer la taille au coefficient qui permet de calculer le prix. Ce n'est pas le but recherché ; la base de données doit être autonome et il s'agit d'une information importante que l'on doit pouvoir retrouver dans la base et qui doit donc y être stockée.

Modification de la base de données

On propose une modification de la base de données pour intégrer cette notion. Une première solution simple serait de remplacer le libellé de la taille par le coefficient qui permet de calculer le prix. On remplace « naine » par « 2/3 », « humaine » par « 1 » et « ogresse » par « 4/3 ». On doit changer le type du champ 'Taille' sans perdre le contenu. Le plus simple est de recréer un champ 'Taille_Prix' et de faire une mise à jour. Les commandes qui modifient le contenu sont les suivantes :

```
Ajout du champ Taille_prix
ALTER TABLE commande
ADD COLUMN Taille_Prix Float;
Création du contenu du champ en fonction du contenu de Taille
UPDATE commande
SET Taille_Prix='2/3'
WHERE Taille='naine' ;
UPDATE commande
SET Taille_Prix='1'
WHERE Taille='humaine' ;
UPDATE commande
SET Taille_Prix='4/3'
WHERE Taille='ogresse' ;
Destruction du champ Taille
ALTER TABLE commande
DROP COLUMN Taille;
```

On peut alors calculer le prix d'une commande directement.

```
SELECT commande.NumCommande, pizza.Prix*commande.Taille_Prix AS Prix_Commande
 FROM commande JOIN pizza ON
 commande.NomPizza=pizza.NomPizza;
```

Réflexion sur la modification apportée et amélioration de la solution

Si le mode de calcul change, par exemple si l'on décide qu'une pizza 'naine' coûte '3/4' du prix normal, il faut alors modifier le(s) coefficient(s) pour toutes les commandes.

On perd également la mention du libellé, plus commode à manipuler et probablement indispensable pour imprimer les factures. On aurait pu conserver le champ 'Taille' avec le libellé pour ne pas perdre cette information, il suffit de ne pas effacer la colonne 'Taille' (dernière commande de la liste précédente). On se trouve alors dans la situation où il existe une dépendance fonctionnelle entre les champs 'Taille' et 'Taille_Prix'. En effet, à une taille correspond un unique prix : la relation 'commande' n'est plus en troisième forme normale et l'on a créé de la redondance.

Pour régler ce problème, il suffit de décomposer la relation « commande » en créant une nouvelle relation « tarification » qui décrira le coefficient en fonction du libellé.

```
Création de la table tarification
CREATE TABLE tarification(
 Taille CHAR(30) PRIMARY KEY,
 Coefficient FLOAT NOT NULL
);
Insertion des données dans la table tarification
INSERT INTO tarification
(Taille, Coefficient)
VALUES ('naine', 2/3) ;
INSERT INTO tarification
(Taille, Coefficient)
VALUES ('humaine', 1) ;
INSERT INTO tarification
(Taille, Coefficient)
VALUES ('ogresse', 4/3) ;
```

On obtient alors les instructions SQL suivantes pour calculer le prix d'une commande. Il est nécessaire de faire une jointure entre les trois tables 'commande', 'pizza' (pour le prix) et 'tarification' (pour le coefficient).

```
SELECT commande.NumCommande, pizza.Prix*tarification.Coefficient AS Prix_Commande
 FROM commande JOIN pizza JOIN tarification ON
 commande.NomPizza=pizza.NomPizza
 AND commande.Taille=tarification.Taille;
```

Implications des modifications apportées au modèle entité-association

Quelles sont les répercussions de cette modification sur le modèle entité-association utilisé jusqu'à présent ? Par un processus inverse, on doit retrouver logiquement le même ensemble de relations. La phrase qui représente l'association entre les entités est la suivante : « Une commande utilise la tarification ». La nouvelle relation introduite provient d'une nouvelle entité associée à l'entité 'commande' :

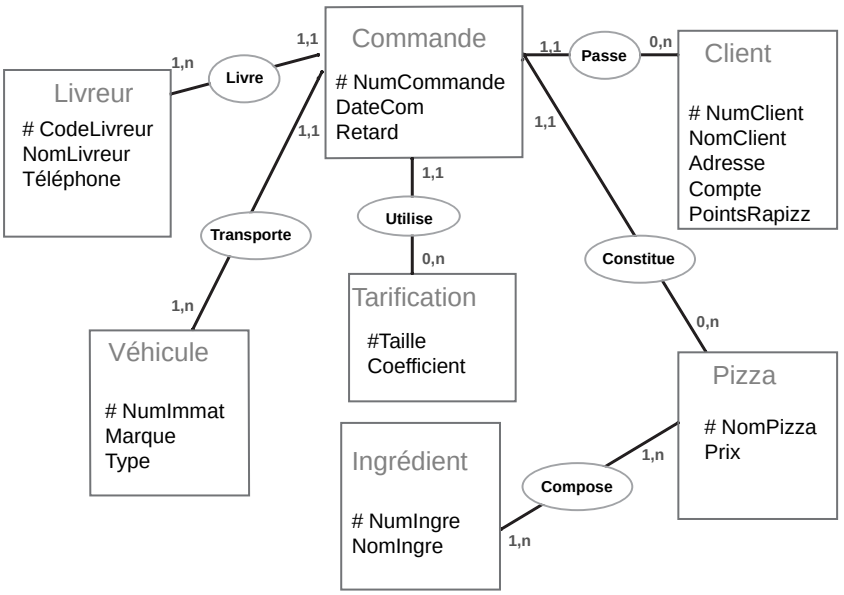
- L'attribut 'Taille' ne se trouve plus dans l'entité 'commande'.
- La nouvelle entité 'Tarification' contient les attributs 'Taille' et 'Coefficient', l'attribut 'Taille' est la clé de cette entité.
- L'entité 'tarification' est associée à l'entité 'commande' par une association que l'on peut nommer 'utilise'.

Cardinalités de l'association 'utilise'.

- La pizza de la commande a une et une seule taille : la cardinalité est de type '1-1'.
- Une tarification peut n'avoir jamais été utilisée pour une commande et plusieurs commandes différentes peuvent avoir la même tarification : la cardinalité est de type '0-n' (voir figure 5.3).

Figure 5.3

Modèle entité-association 'Livraisons de pizzas' avec cardinalités.



On décompose l'association 'emploie' et l'entité 'tarification' :

- L'entité 'tarification' devient une relation dont la clé est 'Taille'
- L'association a une cardinalité '1-1', il s'agit du cas particulier évoqué précédemment, la relation produite par l'association est « aspirée » par la relation créée par l'entité associée avec la cardinalité '1-1', c'est-à-dire 'commande'. La fusion produit le déplacement de l'attribut 'Taille' vers la relation 'commande'.

Finalement, la relation 'commande' reste inchangée même si l'entité dont elle est issue a été modifiée au niveau du modèle entité-association. On a introduit une nouvelle relation 'tarification' comme cela a été fait de manière intuitive précédemment. On obtient l'ensemble de relations suivant :

- client (NumClient, NomClient, Adresse, Compte, PointsRaPizz) ;
- pizza (NomPizza, Prix) ;
- livreur (CodeLivreur, NomLivreur, Téléphone) ;
- véhicule (NumImmat, Marque, Type) ;
- ingrédient (NumIngre, NomIngrédient) ;
- commande (NumCommande, DateCom, Taille, Retard, CodeLivreur, NumImmat, NumClient, NomPizza) ;
- compose (NumIngre, NomPizza) ;
- tarification(Taille, Coefficient).

Épilogue

Cet exemple illustre comment le modèle entité-association peut être remis en cause par nécessité. Le problème de représentation de départ a provoqué une modification du modèle par une démarche inverse de celle utilisée précédemment : on a remis en cause le modèle entité-association à partir de l'ensemble des relations. En revanche, il ne faut pas oublier de modifier le modèle entité-association pour qu'il soit cohérent avec l'ensemble des tables employées dans le SGBD. Les tables sont inutilisables sans modèle descriptif. On peut le constater avec ce système relativement simple qui possède déjà huit tables. Les « allers-retours » entre les différents niveaux de modèles sont fréquents dans la vie d'une base de données.

Remarque

Les requêtes retournent toujours un résultat. Cependant, on ne peut être sûr que ce dernier représente la réponse exacte à la question posée. Il est nécessaire de systématiquement vérifier le résultat sur un jeu de données réduit. De même qu'en programmation, on ne peut prouver qu'un programme réalise correctement ce qu'on lui demande dans tous les cas de figure ; il est difficile d'établir qu'une requête fournit la réponse adéquate lorsque le nombre d'enregistrements devient élevé.

Résumé

Ce chapitre détaille l'ensemble des étapes nécessaires à la réalisation d'une base de données, même les plus fastidieuses. La grande difficulté est de passer d'un énoncé en langage parlé, qui fait généralement suite à un entretien avec les commanditaires et les utilisateurs, à un système utilisable en pratique et efficace. Le flot d'informations recueilli doit être ordonné et structuré. L'exemple choisi est peu complexe, mais il permet d'aborder quelques questions essentielles. On s'aperçoit à cette occasion que l'ensemble des tables utilisées dans le SGBD devient rapidement illisible si l'on ne possède pas un schéma descriptif (modèle entité-association ou UML) correct.

La partie « Interrogation » illustre les catégories d'utilisations et d'interrogations classiques d'une base de données. L'idée est de donner des indications afin d'identifier le type de requête SQL à utiliser en fonction de la question exprimée en langage courant. La démarche suggérée est de toujours décomposer les questions en sous-questions plus simples à résoudre puis de procéder par étapes. Un des exemples aborde ensuite le cas où, au cours de l'utilisation de la base de données, il est impossible de trouver la solution en raison d'une représentation inadéquate. Il est alors nécessaire de remettre en cause le modèle utilisé :

- Soit on modifie le contenu des tables et l'on répercute les modifications sur le modèle descriptif utilisé.
- Soit on modifie le modèle descriptif et l'on réitère les étapes de passage au modèle relationnel.

La seconde méthode est la solution la plus juste d'un point de vue théorique. Mais, en pratique, on utilise une combinaison des deux approches pour obtenir le résultat. Il est essentiel de comprendre qu'il n'y a pas de solution unique en base de données. Le modèle est fréquemment adapté et aucune représentation n'est figée. De même que la réalité peut être envisagée différemment suivant les personnes, il existe souvent plusieurs visions valides.

Exercices

EXERCICE 1 REPRÉSENTATION UML

Énoncé

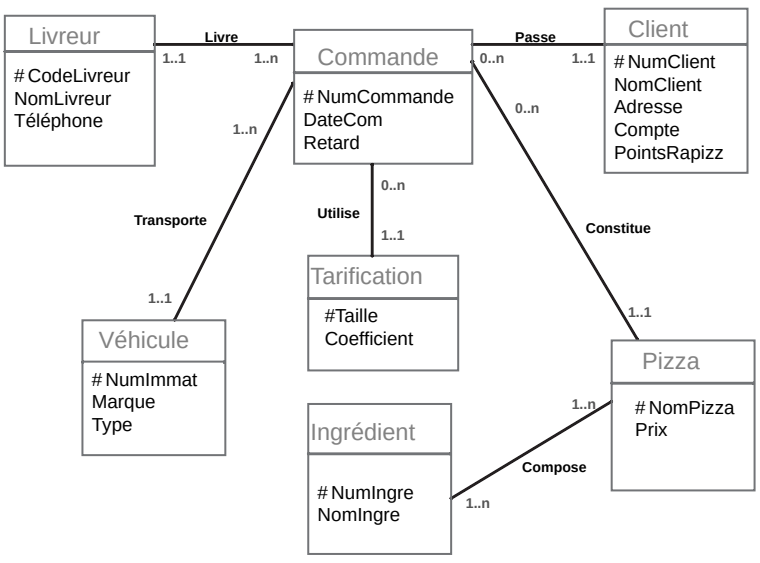
Représentez le modèle entité-association en utilisant le formalisme UML.

Solution

On part du modèle entité-association final, celui auquel on a ajouté l'entité « tarification ». La partie d'UML qui est utilisée est le diagramme de classe ; celui-ci comprend le nom de la classe, la description des attributs et les méthodes associées à la classe. Dans ce cas, on n'utilise pas la partie objet et donc il n'y a pas de méthodes associées aux classes. Une entité est représentée par un diagramme de classe. Avec UML, il est possible d'intégrer les notions de contraintes d'intégrité au niveau du schéma. On peut par exemple spécifier que l'attribut 'retard' de l'entité 'commande' ne pourra prendre comme valeurs que « O » et « N ». Les associations utilisées dans ce message n'ont pas d'attributs. Une différence majeure est que les cardinalités seront positionnées de manière inversée par rapport au modèle entité-association (voir figure 5.4).

Figure 5.4

Modèle UML 'Livraisons de pizzas' avec cardinalités.



EXERCICE 2 CALCULS PAR EXPRESSION SQL

Énoncé

Combien de pizzas ont-elles été livrées en retard ? Quelle est la perte occasionnée par ces retards ?

Solution

Pour calculer le nombre de pizzas livrées en retard, il faut se souvenir d'un point important que l'on a fixé comme « axiome » lors de l'élaboration du modèle : une pizza est associée à une commande et une seule. Une commande ne contient qu'une seule pizza. C'est important : si ce n'était pas le cas, la requête serait beaucoup plus complexe.

Le type de question qui inclut le mot « combien » suggère que l'on va effectuer un comptage sur la table. Il nous suffit de compter le nombre d'enregistrements dont le champ retard est positionné à « Oui ». On a vu lors de la définition de la table 'commande' que le contenu du champ retard a été normalisé et restreint par une contrainte d'intégrité aux valeurs « N » et « O ». Ces réflexions conduisent à la simple requête suivante :

```
SELECT COUNT(*) AS NombreRetard
 FROM commande
 WHERE retard='O';
```

On compte le nombre de lignes de la table commande dont le champ commande est égal à « O ». On ne spécifie pas de nom de champ dans la fonction « COUNT » de SQL, car il s'agit d'un calcul indépendant d'un champ donné.

Pour connaître la perte occasionnée, on calcule la somme du prix de chaque commande. Le modèle a été modifié pour pouvoir calculer directement cette information. On a besoin du prix de base de la pizza qui se trouve dans la table 'pizza' et du coefficient qui se trouve dans la table 'tarification'. On effectue une jointure sur ces trois tables.

```
SELECT SUM(pizza.prix*tarification.coefficient) AS PerteRetard
 FROM commande JOIN pizza JOIN tarification
 ON commande.NomPizza=pizza.NomPizza AND commande.Taille=tarification.Taille
 WHERE commande.retard='O';
```

La fonction SQL SUM a besoin du contenu précis du champ concerné par le calcul. Ici, il s'agit d'une expression constituée à partir du prix et du coefficient. Il est préférable de préfixer les noms de champ par le nom de la table d'où ils proviennent pour éviter les ambiguïtés et faciliter la lecture ultérieure de la requête. Les ambiguïtés sont signalées par le SGBD qui refuse d'exécuter la requête.

On aurait pu écrire les jointures en utilisant classiquement un produit cartésien et une sélection. Dans ce cas, les expressions qui servent à la jointure sont mélangées à celles qui servent à faire la sélection dans la clause WHERE. De plus, la requête est effectuée de manière beaucoup moins efficace par le SGBD ; cela est nettement perceptible lorsque l'on dispose de tables de taille importante.

```
SELECT SUM(pizza.prix*tarification.coefficient) AS PerteRetard
 FROM commande, pizza, tarification
 WHERE commande.NomPizza=pizza.NomPizza AND commande.Taille=tarification.Taille
 AND commande.retard='O';
```

Les deux tables « résultat » sont des tables qui possèdent une seule ligne et une seule colonne. On peut utiliser ces valeurs par exemple pour faire une comparaison (voir l'un des exercices suivants) en les associant par un produit cartésien à la table sur laquelle on veut effectuer la comparaison.

EXERCICE 3 CODE SQL ET SIGNIFICATION

Énoncé

Cette requête SQL donne-t-elle un résultat ? Si oui, que signifie-t-il ?

```
SELECT client.NomClient, livreur.NomLivreur
 FROM livreur JOIN client
 ON livreur.CodeLivreur=client.NumClient ;
```

Imaginez à quel type de question a voulu répondre la personne qui a fait cette requête.

Solution

La requête retourne un résultat, car la syntaxe est correcte et le type des champs sur lesquels on a effectué la jointure sont de type compatible et contiennent des valeurs communes. Bien évidemment, le résultat n'a aucun sens d'un point de vue la réalité. Le schéma entité-association montre que les liens entre les tables 'client' et 'livreur' passent par la table 'commande' et les associations 'livre' et 'passe'. Il s'agit du cas typique qui illustre le fait qu'une requête donne toujours un résultat, même s'il n'a aucun sens.

En aucun cas, cette requête ne permettrait d'effectuer des recherches de corrélation entre le livreur et le client. Si l'on cherche à afficher le nom du client et le nom du livreur correspondant par commande, on doit utiliser la table 'commande' même si l'on ne projette aucun champ de la table 'commande'. Pour faciliter la lecture et repérer d'éventuelles corrélations, on ordonne par noms de clients.

```
SELECT client.NomClient, livreur.NomLivreur
 FROM livreur JOIN commande JOIN client
 ON livreur.CodeLivreur=commande.CodeLivreur
    AND client.NumClient= commande.NumClient
ORDER BY client.NomClient;
```

Pour savoir quels clients sont toujours livrés par le même livreur, la démarche serait plus complexe.

EXERCICE 4 AGRÉGATS ET SÉLECTION

Énoncé

Donnez le chiffre d'affaires par pizza vendue. On ne tient pas compte à ce niveau des pizzas gratuites obtenues grâce aux points de fidélité ou en raison d'un retard de livraison.

Solution

La méthode conseillée dans ce chapitre est d'aborder la question en décomposant le problème. Pour calculer le chiffre d'affaires, il faut d'abord regrouper les commandes par pizza et ensuite effectuer le calcul. La notion de regroupement suggère l'emploi des agrégats. Le nom de la pizza est directement accessible dans la table 'commande'. Pour vérifier combien de commandes ont été passées par pizza, on les compte.

```
SELECT commande.NomPizza, COUNT(*) AS NombreCommande
 FROM commande
 GROUP BY commande.NomPizza;
```

Pour calculer le prix, on a besoin du prix de base qui se trouve dans la table 'pizza' et du coefficient qui se trouve dans la table 'tarification'.

```
SELECT commande.NomPizza, COUNT(*) AS NombreCommande, SUM(pizza.prix*tarifica-
tion.coefficient) AS TotalCommande
 FROM commande JOIN pizza JOIN tarification
 ON commande.Taille=tarification.Taille AND commande.NomPizza=pizza.NomPizza
 GROUP BY commande.NomPizza
 ORDER BY TotalCommande;
```

On trie le résultat en utilisant le champ calculé 'TotalCommande'. L'opération de tri se fait donc après les opérations de jointures, d'agrégation et de calculs. Il est possible de réaliser une sélection sur le résultat final de cette opération en ne considérant que les pizzas dont le chiffre d'affaires dépasse le nombre « 200 ». On n'emploie pas dans ce cas le mot clé WHERE qui sert à réaliser les sélections sur une table standard. On utilise le mot clé HAVING qui fait une sélection sur le résultat des calculs sur les agrégats. Cette sélection se fait, de même que le tri, à la fin de l'opération sur la table « résultat » finale.

```
SELECT commande.NomPizza, COUNT(*) AS NombreCommande, SUM(pizza.prix*tarifica-
tion.coefficient) AS TotalCommande
```

```
FROM commande JOIN pizza JOIN tarification
ON commande.Taille=tarification.Taille AND commande.NomPizza=pizza.NomPizza
GROUP BY commande.NomPizza
HAVING TotalCommande > 200
ORDER BY TotalCommande;
```

En revanche, si l'on avait voulu éliminer du résultat les pizzas dont le prix de vente n'est pas assez élevé considérant que cela fausse le résultat, le mode de sélection ne serait pas le même. Il faudrait réaliser une sélection sur l'ensemble de départ avant d'effectuer les agrégats et les calculs ; dans ce cas, on utiliserait le mot clé WHERE.

```
SELECT commande.NomPizza, COUNT(*) AS NombreCommande, SUM(pizza.prix*tarifica-
tion.coefficient) AS TotalCommande
 FROM commande JOIN pizza JOIN tarification
 ON commande.Taille=tarification.Taille AND commande.NomPizza=pizza.NomPizza
 WHERE pizza.prix > 10
 GROUP BY commande.NomPizza
ORDER BY TotalCommande;
```

Dans le premier cas, la sélection est faite *a fortiori* ; dans le second, *a priori*.

EXERCICE 5 REQUÊTES COMBINÉES

Énoncé Quel est le nom du livreur qui a le plus de retard ?

Solution

On décompose le problème :

- calcul du nombre de retards par livreur ;
- calcul du maximum de ces retards ;
- sélection du nom du livreur qui a le maximum de retard.

Calcul du nombre de retards par livreur. Le mot « par » suggère comme précédemment l'emploi d'agrégats sur lesquels on utilise la fonction de comptage.

```
CREATE TEMPORARY TABLE requete1
SELECT commande.CodeLivreur, COUNT(*) AS NombreRetard
FROM commande
GROUP BY commande.CodeLivreur
ORDER BY NombreRetard
;
```

Calcul du maximum de ces retards. On utilise la fonction SQL MAX sur le contenu de la table précédente.

```
CREATE TEMPORARY TABLE requete2
SELECT MAX(NombreRetard)AS MaxRetard
FROM requete1
;
```

Sélection du nom du livreur qui a le maximum de retard. Il est demandé d'afficher le nom du livreur, disponible uniquement dans la table 'livreur'. On fait classiquement une jointure pour le récupérer. On doit également disposer de la valeur du maximum de retard que l'on trouve dans la table temporaire 'requete2'. On l'associe à l'opération sur les deux autres tables par un produit cartésien. Cela ne change pas le nombre de lignes du résultat car la table 'requete2' n'a qu'une ligne.

Il s'agit d'un cas où l'on effectue dans la même requête un produit cartésien et une jointure.

```
SELECT livreur.NomLivreur
FROM requete1 JOIN livreur ON livreur.CodeLivreur=requete1.CodeLivreur, requete2
WHERE requete1.NombreRetard=requete2.MaxRetard
;
```

Avec un peu d'expérience, on s'aperçoit qu'il s'agit d'une requête semblable à celle que l'on présente dans le chapitre pour connaître les clients qui commandent plus que la moyenne.

EXERCICE 6 SIMPLE SÉLECTION OU JOINTURE EXTERNE ?

Énoncé

Quel est le nom du livreur qui n'est jamais en retard ?

Solution

La tournure de phrase « qui n'a jamais » semble suggérer l'emploi d'une jointure externe, comme c'était déjà le cas dans le chapitre où on a déterminé les véhicules n'ayant jamais servi. Cependant, cette jointure externe sert à déterminer les entités non associées dans une jointure. Ici, on peut obtenir l'information du retard directement dans la table 'commande'. Si l'on considère les résultats de l'exercice précédent, on a calculé le nombre de retards par livreur par la requête suivante.

```
CREATE TEMPORARY TABLE requete1
SELECT commande.CodeLivreur, COUNT(*) AS NombreRetard
FROM commande
GROUP BY commande.CodeLivreur
ORDER BY NombreRetard
;
```

Un livreur n'aura jamais été en retard si le nombre de retards ainsi calculé est égal à zéro.

```
SELECT requete1.CodeLivreur
FROM requete1
WHERE requete1. NombreRetard=0
;
```

On cherche le nom du livreur et non pas son code. Il y a deux manières de procéder :

- Soit on fait une jointure sur la table 'livreur' pour la première requête de manière à inclure le nom du livreur dans la table résultat.

```
CREATE TEMPORARY TABLE requete1
SELECT commande.CodeLivreur, livreur.NomLivreur, COUNT(*) AS NombreRetard
 FROM commande JOIN livreur
 ON commande.CodeLivreur=livreur.CodeLivreur
GROUP BY commande.CodeLivreur
ORDER BY NombreRetard

SELECT requete1.NomLivreur
FROM requete1
WHERE requete1. NombreRetard=0
;
```

- Soit on fait la jointure lors de la seconde requête.

```
CREATE TEMPORARY TABLE requete1
SELECT commande.CodeLivreur, COUNT(*) AS NombreRetard
FROM commande
GROUP BY commande.CodeLivreur
ORDER BY NombreRetard
;
SELECT livreur.NomLivreur
FROM requete1 JOIN livreur ON requete1.CodeLivreur=livreur.CodeLivreur
WHERE requete1. NombreRetard=0
;
```

Il est possible de regrouper ces deux requêtes en une seule. La stratégie de regroupement dépendra du SGBD employé.

Le cas serait différent si l'on cherchait à identifier les noms des livreurs qui n'ont jamais effectué de livraison.

L'action de livraison est représentée par l'association 'livre' dans le modèle entité-association. Cette association, puisque de cardinalité '1-1', a « disparu » en tant que table et a été intégrée dans la relation 'commande' issue de l'entité 'commande'.

En résumé, l'information n'est disponible que par une jointure entre les tables 'commande' et 'livreur'. L'idée ici est de trouver les codes des livreurs présents dans la table 'livreur' mais pas dans la table 'commande'. D'après notre définition du modèle entité-association, ce ne devrait pas être possible puisque la cardinalité du point de vue (du côté) de l'entité 'livreur' est de type '1-n' : un livreur a au moins livré une pizza et est susceptible d'en livrer plusieurs. On emploie alors une jointure externe qui nous permet d'inclure dans le résultat les lignes n'ayant pas de correspondance dans la table 'commande'.

```
SELECT *
FROM livreur LEFT JOIN commande
 ON livreur.CodeLivreur=commande.CodeLivreur
;
```

Pour sélectionner ceux qui n'auraient pas effectué de livraison, on choisit une ligne dont un champ issu de la table 'commande' est vide (a une valeur nulle en SQL). On projette sur le champ 'NomLivreur' de la table 'commande' qui était l'information demandée.

```
SELECT livreur.NomLivreur
FROM livreur LEFT JOIN commande
 ON livreur.CodeLivreur=commande.CodeLivreur
 WHERE commande.CodeLivreur IS NULL
;
```

EXERCICE 7 MISE À JOUR DE LA BASE

Énoncé

On veut mettre à jour le solde du compte d'un client après chaque commande. Pour ce faire, il est nécessaire de recourir à un langage de programmation ou à des notions qui dépassent le cadre de cet ouvrage ; on s'intéresse simplement aux différentes requêtes à prévoir pour réaliser cette opération. Pour l'instant, on considérera que l'on crée la requête de mise à jour sans se préoccuper des aspects de solde (négatif ou positif) du compte.

Solution

On calcule le prix de la commande comme on l'a vu précédemment. La commande est identifiée par son numéro de commande, ici 60. On aura également besoin du numéro de client pour faire la mise à jour.

```
CREATE TEMPORARY TABLE requete1
SELECT commande.NumClient, pizza.Prix*tarification.Coefficient AS Prix_Commande
 FROM commande JOIN pizza JOIN tarification ON
 commande.NomPizza=pizza.NomPizza
 AND commande.Taille=tarification.Taille
 WHERE NumCommande=60 ;
```

La requête de mise à jour serait la suivante, on récupère les champs prix de la commande de la première requête pour effectuer la mise à jour du solde du compte du client dans la table 'client'.

```
UPDATE client,requete1
 SET Compte=Compte-requete1.Prix_Commande
 WHERE client.NumClient=requete1.NumClient ;
```

Il est prévu de pouvoir lancer dans un SGBD une requête au moment où l'on effectue une opération : on appelle cela des *triggers* ou procédures stockées. Typiquement, il faudrait lancer cette requête chaque fois que l'on crée une commande.

EXERCICE 8 ÉVOLUTION DE LA BASE DE DONNÉES

Énoncé

On veut pouvoir calculer les prix des pizzas hors taxes et toutes taxes. Que faut-il modifier dans la base de données selon que le taux est considéré comme fixe ou susceptible de varier ?

Solution

Pour traiter ce genre de cas, on doit se rappeler que l'on ne stocke jamais une donnée qui peut être calculée. Par conséquent, on ne stocke pas les prix hors taxes et toutes taxes de chaque pizza. On suppose que le prix stocké dans la table 'pizza' est le prix hors taxes. Si le taux est fixe, on peut l'utiliser pour les calculs de prix toutes taxes. On affiche le contenu de l'expression qui permet de calculer le prix toutes taxes.

```
SELECT NomPizza, Prix*(1+19.6) AS Prix_Ttc
FROM pizza ;
```

Si le taux n'est pas fixe, il faut l'inclure dans le système d'information ; on crée une table 'taxe' avec une ligne et une colonne, qui contiendra uniquement le taux.

```
CREATE TABLE taxes (
Taux  FLOAT
);
INSERT INTO taxes
(Taux)  VALUES (19.6) ;
```

On récupère la valeur du taux comme on l'a fait précédemment, en faisant un produit cartésien avec la table 'taxes'.

```
SELECT NomPizza, Prix*(1+Taux) AS Prix_Ttc
FROM pizza, taxes ;
```

Dans ce second cas, il faudrait pour être complet ajouter l'entité 'taxes' au modèle entité-association. Elle ne serait liée à aucune entité par une association.

EXERCICE 9 AUTRE EXEMPLE COMPLET

Énoncé

Xavier S. possède deux établissements hôteliers dans le Limousin : « Le Saint-Bob » à Saint-Robert et « Le Saint-Ségur » à Ségur-le-Château. Il désire automatiser la gestion des réservations. Les chambres proposées dans les deux établissements sont de trois types : simples (50 €/jour), doubles (65 €/jour) et royales (150 €/jour). Dans chacun des deux hôtels, les chambres sont numérotées en fonction de l'étage : les chambres en rez-de-chaussée portent les numéros 001, 002… ; les chambres du premier étage portent les numéros 100, 101… ainsi de suite. Les réservations sont centralisées à Saint-Robert au 0800 800 800 et un employé est chargé de répondre au téléphone pour les deux établissements. Lorsqu'un client réserve une chambre, s'il n'est pas référencé, on lui demande son nom, son prénom, son numéro de téléphone ainsi que son adresse.

Énoncé

1. Réalisez le modèle conceptuel des données, spécifiez les cardinalités.
2. Déduisez-en le modèle entité-association correspondant.
3. Créez les tables correspondantes (commande SQL 'create table'), en spécifiant les clés uniques. Indiquez les contraintes d'intégrité pour ces tables.
4. Donnez la requête permettant de lister les réservations effectuées en 2006.

Solution

Les numéros de chambres ne peuvent pas constituer un identifiant de l'entité 'Chambre', dans la mesure où les deux hôtels possèdent des chambres ayant le même identifiant. En conséquence, il convient d'ajouter un identifiant unique supplémentaire à l'entité 'Chambre'. La période de réservation modélisée à l'aide d'une date de début et d'une date de fin dépend fonctionnellement de la chambre et du client. Le nombre de jour sera déduit à partir de ces données à l'aide d'un traitement informatique afin de procéder à la facturation du séjour. Le type de chambre déterminant le prix de la chambre, une entité supplémentaire 'Type' doit être créée.

1. Modèle conceptuel

   Hôtel(ID_Hotel, Nom_hotel, CP, Ville)

   Chambre(ID_Chambre, Num_chambre)

   Type(ID_Type, Libelle, Prix)

   Reserver(Date_debut, Date_Fin) entre Client et Chambre (1,n – 0,n)

   Composer() entre Chambre et Hotel (1,1 – 1,n)

   Correspondre() entre Chambre et Type (1,1 – 0,n)

2. Modèle relationnel

   Hotel(ID_Hotel, Nom_hotel, CP, Ville)

   Chambre(ID_Chambre, Num_chambre, ID_hotel, ID_Type)

   Type(ID_Type, Libelle, Prix)

   Reserver(Code_client, ID_Chambre, Date_debut, Date_fin,)

   Client(Code_client, Nom_client, Prenom_client, Num_tel, Adresse, CP, Ville)

3. Code SQL de création des tables

```
CREATE TABLE HOTEL
ID_HOTEL INT PRIMARY KEY,
NOM_HOTEL VARCHAR(20) NOT NULL,
CP VARCHAR(5) NOT NULL,
VILLE VARCHAR(50) NOT NULL
) ;


CREATE TABLE CHAMBRE(
ID_CHAMBRE INT PRIMARY KEY,
NUM_CHAMBRE INT NOT NULL,
ID_HOTEL INT REFERENCES HOTEL(ID_HOTEL),
ID_TYPE INT NOT NULL REFERENCES TYPE(ID_TYPE)
) ;


CREATE TABLE TYPE(
ID_TYPE INT PRIMARY KEY,
LIBELLE VARCHAR(50) NOT NULL,
PRIX FLOAT(10,2),
CHECK LIBELLE IN ('SIMPLE','DOUBLE','ROYALE')
) ;
```

```
CREATE TABLE RESERVER(
CODE_CLIENT VARCHAR(9) NOT NULL REFERENCES CLIENT(CODE_CLIENT,
ID_CHAMBRE INT NOT NULL REFERENCES CHAMBRE(ID_CHAMBRE),
DATE_DEBUT DATE, DATE_FIN DATE
) ;


CREATE TABLE CLIENT(
CODE_CLIENT VARCHAR(9) PRIMARY KEY,
NOM_CLIENT VARCHAR(50) NOT NULL,
PRENOM_CLIENT VARCHAR(50) NOT NULL,
NUM_TEL VARCHAR(10) NOT NULL,
ADRESSE VARCHAR(250),
CP VARCHAR(5),
VILLE VARCHAR(50) NOT NULL
) ;
```

4. Code SQL permettant de lister les réservations de 2006

```
SELECT *
FROM `RESERVER`
WHERE `DATE_DEBUT` >= '01/01/2006 ' AND `DATE_FIN` <= '31/12/2006'
;
```

Chapitre 6

Préservation des données

Ce chapitre traite de la sécurité des données qui est une notion fondamentale des bases de données. En effet, le pire pour une base de données est la perte ou la modification de données. Des problèmes sérieux peuvent être provoqués de manière intentionnelle ou accidentelle, mais le résultat obtenu est le même.
L'activité des organisations repose sur les données : il convient par conséquent de les protéger, mais également d'en assurer la disponibilité permanente. On distinguera plusieurs niveaux de granularité quant aux recommandations générales concernant la sécurité des données :

- la protection de l'accès à la machine d'un point de vue physique ou réseau, les aspects de durée de vie du système ainsi que la politique de sauvegarde ;
- la politique de restriction d'accès aux données du SGBD par des comptes et l'utilisation des vues ;
- la capacité des outils du SGBD à protéger les opérations sur les données, à les restaurer et à revenir en arrière, comme pour les transactions.

On introduit en fin de chapitre un outil complémentaire : le « trigger ». Les triggers (ou déclencheurs) sont utilisés pour forcer une bonne gestion du contenu des données lors des opérations d'insertion, de mise à jour ou de suppression de données.

1 Contrôle d'accès et sauvegarde

Toute réflexion autour de la sécurité doit se concevoir de manière globale en utilisant en premier lieu le bon sens : il est inutile d'installer une porte blindée inviolable sur une palissade en bois. Les recommandations qui sont énoncées dans cette section sont applicables à tous les systèmes informatiques, en particulier à ceux qui sont connectés à un réseau ou situés dans un milieu qualifié d'« hostile ».

On décrit ici plusieurs types de préoccupations :

- l'accès physique à la machine ;
- l'accès distant à la machine à travers le réseau ;
- les aspects de pérennité de l'ensemble machine, système d'exploitation et SGBD ;
- les sauvegardes, la redondance des données et leur accessibilité.

1.1 Accès physique à la machine

Cette mesure de sécurité informatique élémentaire est très souvent négligée. Procurer un accès physique à la machine permet en général de contourner toutes les protections liées au système d'exploitation ou au SGBD que l'on aborde dans ce chapitre. Il suffit de démarrer une machine avec un autre système d'exploitation présent sur un CD ou une clé USB par exemple pour obtenir un accès au disque et ainsi aux données associées au SGBD. La première précaution à prévoir consiste à protéger le BIOS (*Basic Input Output System*) ou l'équivalent de la machine pour l'empêcher de démarrer sur un autre disque que celui qui contient le « bon » système d'exploitation.

Le fait de pouvoir accéder directement à la machine permet également à une personne animée de mauvaises intentions d'empêcher l'accès aux données en détruisant physiquement le disque sur lequel se trouvaient les fichiers ou même la machine.

Par conséquent, la ou les machines qui contiennent le SGBD et les fichiers de données doivent se trouver dans des pièces dont l'accès est, au minimum, contrôlé. Dans le même ordre d'idée, l'alimentation électrique ne doit pas être accessible. On a tous entendu ou vécu une histoire de machines ou d'éléments réseaux débranchés par erreur pour effectuer le ménage, sans parler de la malveillance. Comme on peut le constater, les précautions concernant l'accès physique à la machine ne sont pas toujours d'ordre purement technique ; elles relèvent souvent du simple bon sens commun.

1.2 Accès à la machine par le réseau

L'accès à la machine par le réseau est devenu une porte d'entrée privilégiée pour les personnes malintentionnées, car rares sont les serveurs qui ne sont pas connectés aujourd'hui. On distingue ainsi principalement deux aspects de problèmes de sécurité :

- les attaques « internes » qui proviennent de personnes ayant accès à la machine par un compte standard associé à un mot de passe correct ;
- les attaques « externes » qui se font en profitant d'une faille du système d'exploitation ou d'une des applications installées sur la machine.

Dans les deux cas, on peut être confronté à un problème de destruction de données par simple effacement de fichiers de données. Cette opération d'effacement sera rendue plus ou moins facile suivant les systèmes employés. Dans le premier cas, il est en principe

impossible qu'un simple utilisateur efface les fichiers des autres, mais certains systèmes sont plus « perméables » que d'autres. Dans le second cas, si l'on peut devenir administrateur sur la machine concernée du point de vue du système d'exploitation, tout est possible. S'il s'agit d'un cas de malveillance, les dégâts peuvent être considérables.

Les protections dans ce domaine ne dépendent pas spécifiquement de l'administrateur du SGBD, mais aussi de l'administrateur système et réseau de l'environnement dans lequel se trouve la machine qui héberge le SGBD. Le système et les applications présentes sur la machine doivent être maintenus et mis à jour régulièrement pour éviter les problèmes. L'administrateur système doit également tenir à jour les comptes utilisateurs qui possèdent les droits d'accès à la machine et vérifier la qualité des mots de passe associés. Classiquement, une intrusion se fait en utilisant un compte « oublié », créé par exemple pour un besoin ponctuel, dont le mot de passe est assez simple pour être deviné. Le SGBD lui-même est une application parmi les autres ; il peut présenter des failles et doit faire partie du programme des mises à jour critiques. Cette tâche est généralement effectuée par l'administrateur du SGBD, qui doit également tenir à jour les comptes d'utilisateurs spécifiques du SGBD disposant de droits d'accès à la machine.

En résumé, le système sur lequel est hébergée la base de données doit faire l'objet d'un suivi permanent à tous les niveaux. Il n'est pas raisonnable d'installer un SGBD sur une machine dont le système ne peut être mis à jour ou dont les mises à jour sont disponibles trop tardivement. Toute application présente sur la machine peut recéler des failles et être le maillon faible de l'ensemble. Les informations de vulnérabilité et les correctifs sont fournis par les éditeurs de ces applications. Dans notre cas, on doit porter une attention particulière à une application importante : le SGBD. La réactivité des éditeurs constitue un critère décisif pour choisir un système et un SGBD. D'un point de vue humain, la qualité de l'administration du système et du réseau sur lequel se trouve la machine peut être également un élément du choix.

1.3 Pérennité du système

Une base de données se conçoit généralement pour de nombreuses années d'utilisation. Il est donc essentiel, avant de choisir une machine, son système d'exploitation ainsi que le logiciel de SGBD, de prendre en considération leurs pérennités respectives. Le choix peut alors se porter sur des ensembles moins performants mais que l'on considère comme plus viables en termes de durée de vie.

Voici quelques points à considérer :

- **Le suivi de la machine elle-même.** Par exemple, la durée de la garantie matérielle, la possibilité de retrouver les pièces etc.
- **La durée de vie du système d'exploitation.** Un système qui n'est plus mis à jour par son éditeur et qui présente des trous importants de sécurité se révèle un très mauvais choix. La plupart des éditeurs continuent à assurer une maintenance minimale pour les anciens systèmes, mais ce n'est pas systématique.
- **L'évolutivité du SGBD.** Un SGBD propriétaire qui n'offre pas les échanges de données avec d'autres logiciels handicape la migration vers un autre SGBD. De même que pour les points précédents, un SGBD qui présente des failles importantes de sécurité d'accès ou de fonctionnement et qui n'est plus mis à jour par l'éditeur constitue un problème potentiel qu'aucune autre précaution ne saurait combler.

Cette projection dans l'avenir n'est pas toujours évidente compte tenu du nombre de paramètres à prendre en compte. Il est difficile de prédire qu'un constructeur ou un éditeur ne

disparaîtra pas dans les années à venir. Il est important d'envisager dès la conception l'éventualité d'une migration vers un autre SGBD, éventuellement hébergé par un autre système.

1.4 Sauvegarde, redondance et accessibilité des données

Sauvegarde des données

Des **copies de sauvegarde** des données contenues dans la base, mais aussi des métadonnées (dictionnaire de données) du SGBD, doivent être réalisées régulièrement. Outre les problèmes de malveillance, un incendie ou une inondation peuvent réduire à néant des années de travail. Dans la mesure du possible, les sauvegardes ne doivent pas être stockées dans le même endroit que la machine. L'idéal étant de disposer d'une armoire de stockage adaptée qui soit ignifugée et située dans un endroit à l'abri des inondations. Si ce stockage se trouve dans un lieu différent, c'est encore mieux.

Compte tenu du volume des données utilisé actuellement, la sauvegarde peut poser des problèmes de taille de support physique de stockage. À cet effet, on utilise fréquemment des robots de sauvegarde spécialisés, manipulant des bandes que l'on peut regrouper pour former un espace de stockage suffisant. Les disques regroupés dans un dispositif autonome (type RAID) sont également très employés pour les sauvegardes en raison de leur prix relativement abordable. Ils apportent de plus une certaine sécurité liée à des mécanismes de redondance, qui leur assure un fonctionnement sans pertes de données en cas de panne de l'un des disques.

Redondance des données

Un procédé plus efficace et plus sûr consiste à disposer de copies de la base en plusieurs endroits géographiquement distincts. La plupart des SGBD sont capables d'effectuer des mises à jour sur des copies de la base de données situées sur d'autres machines *via* le réseau. La fréquence de ces mises à jour va dépendre de celle de modification des données. Cette méthode permet de plus d'améliorer la **disponibilité** des données :

- Si l'un des serveurs est inaccessible, un autre peut prendre le relais.
- Il est possible ainsi de répartir la charge sur plusieurs serveurs.

On obtient une **redondance** de machines et donc des données. Cela peut sembler paradoxal dans la mesure où une grande partie de l'ouvrage est consacrée à la suppression de la redondance, mais le concept mis en œuvre ici est différent. Les données sont simplement dupliquées pour se prémunir contre leur perte et augmenter leur disponibilité. Ce dispositif est complexe à gérer d'un point de vue du réseau et nécessite de disposer de plusieurs hébergements géographiquement distants (voir figure 6.1).

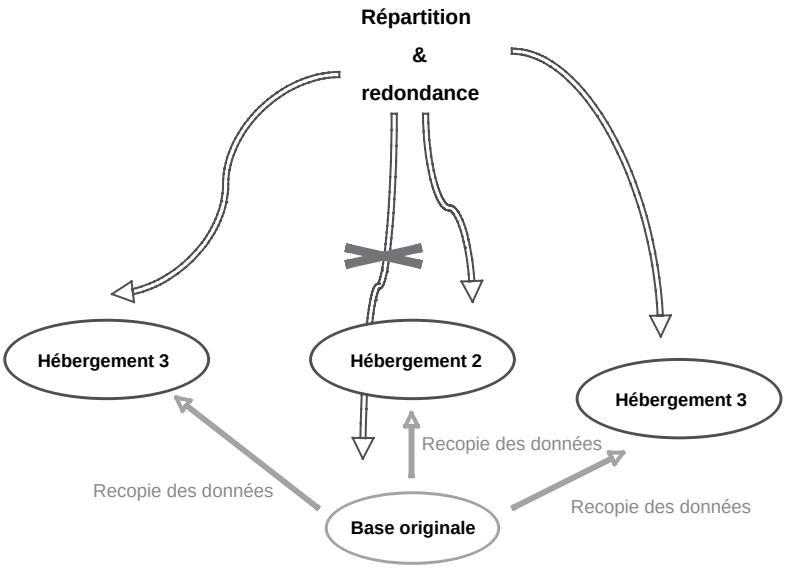


Figure 6.1

Redondance et répartition de charge.

2 Limitations d'accès au SGBD

Après avoir résolu les problèmes généraux d'accès à la machine, abordés dans la section précédente, il convient de restreindre l'accès au SGBD. Ces restrictions concernent les autorisations accordées ou non sur les différents éléments que contient le SGBD, mais aussi permettent de contrôler l'accès général au SGBD.

Ces opérations de gestion sont effectuées typiquement par l'administrateur de la base de données, qui définira des identifiants et leur affectera des droits, de manière totalement indépendante du système d'exploitation. Ces informations seront stockées dans le dictionnaire de données du SGBD. Il est possible d'accéder au SGBD, éventuellement à distance, sans posséder de compte sur la machine qui l'héberge. L'administrateur du SGBD doit donc gérer avec le plus grand soin ces identifiants, afin qu'ils deviennent pas un moyen de pénétrer dans le système d'exploitation en cas de faille dans le SGBD.

2.1 SQL et la gestion des utilisateurs

Les instructions qui permettent d'affecter les droits sont définies dans la norme SQL. Comme pour les autres éléments de la norme, les éditeurs ne les ont pas toujours intégrées totalement dans leurs produits. À l'inverse, ils ont souvent ajouté des instructions qui ne figurent pas dans la norme pour gérer ces droits. Classiquement, pour protéger une ressource, on commence par en interdire l'accès à tous. Puis on décrit ceux qui sont autorisés à l'utiliser (utilisateurs). On associe ensuite à chaque utilisateur un identifiant (mot de passe) pour éviter les emprunts d'identité.

Dans le principe, SQL ne considère pas réellement la notion d'utilisateur pour le SGBD tel qu'on le conçoit au sens d'un système d'exploitation. On peut envisager l'utilisation de plusieurs identifiants associés au même nom : cela correspondrait grossièrement à la possibilité d'utiliser des mots de passe spécifiques selon la ressource. Il s'agit donc d'une logique différente de celle qui consiste à associer strictement un mot de passe à un « compte » et à lui procurer des droits sur les ressources. La norme SQL considère que c'est l'objet à

protéger qui est au centre du processus : les droits d'utilisation de cet objet sont ensuite affectés à des couples de types (nom, identifiant).

Le SGBD stocke les informations suivantes pour chaque objet qu'il contient :

- **Le type de droit.** Sélection, création, destruction…
- **L'objet sur lequel s'appliquent les droits.** Une base de données, une table, une vue…
- **Le nom de l'utilisateur.**
- **L'identifiant, au sens du mot de passe.**
- **Le donneur des droits.**

Remarque

En pratique, la plupart des SGBD intègrent une instruction pour créer un utilisateur et lui associer un identifiant qui reste unique. Même si elle n'est pas définie réellement dans la norme SQL, elle est fréquemment de la forme :

```
CREATE USER <type de droit> IDENTIFIED BY <identifiant> ;
```

Cette instruction met à jour le dictionnaire de données du SGBD. Ce dernier utilise les informations de l'utilisateur pour authentifier la connexion au SGBD. Là encore, le mode de connexion dépend du SGBD utilisé. Par extension, celui-ci fournira en général une instruction du type DROP USER pour détruire l'utilisateur créé par la précédente commande.
Comme les informations du dictionnaire de données sont le plus souvent stockées dans des bases de données, on peut également les manipuler avec des instructions SQL classiques de types INSERT et DELETE. La spécificité de l'instruction CREATE USER est de posséder une instruction de cryptage du mot de passe.

Compte tenu de la remarque précédente, les éditeurs intègrent souvent la notion plus classique d'utilisateur dans un SGBD. Ce dernier permet en général de distribuer des autorisations à un ensemble d'utilisateurs qui constituent ainsi un **groupe**. Les tâches de gestion en sont facilitées, mais les groupes, qui représentent l'organisation de l'entreprise, sont parfois complexes à gérer. L'affectation de droits à une hiérarchie de groupes et sous-groupes peut relever du casse-tête.

Afin de résoudre ce problème, on dispose d'un autre modèle de distribution des droits : le **rôle**. Celui-ci désigne un assortiment de droits que l'on désire affecter à un objet du SGBD. C'est une vue inversée où les utilisateurs prennent le(s) rôle(s) dont ils ont besoin pour pouvoir travailler sur les objets du SGBD qui les concernent. L'instruction pour créer un rôle est la suivante :

```
CREATE ROLE consultation_seulement ;
```

L'affectation des droits à un rôle et la distribution de rôles à des utilisateurs sont présentées à la section suivante. La gestion des rôles, même s'ils font partie de la norme SQL, n'est pas proposée par tous les SGBD. Il en est de même pour les groupes d'utilisateurs ou tout simplement de la notion d'utilisateur qui n'existe pas toujours dans le SGBD.

En résumé, il n'y a pas de règle générale de gestion des autorisations de connexion au SGBD. L'initiative en est laissée à l'éditeur du SGBD. Cet aspect peut provoquer des soucis lors de la migration vers un autre SGBD.

2.2 SQL ET LA GESTION DES DROITS

Une fois que l'utilisateur est connecté au système, il convient de gérer son accès aux données. Une pratique courante pour les SGBD est que tout nouvel objet créé – une base de données par exemple – est par défaut inaccessible à tous. Les droits sont distribués ensuite à des utilisateurs, des groupes ou des rôles. Il existe ensuite une hiérarchie entre les différents objets du SGBD : une table est comprise dans une base de données ; un champ est compris dans une table, etc. Afin de simplifier la gestion, la plupart des SGBD permettent l'affectation de droits à un objet associé à tous les sous-objets qui constituent une branche de la hiérarchie.

Droits sur les objets du SGBD

Avec SQL, il est possible de spécifier des droits essentiellement sur les opérations de manipulation de tables (ou de vues considérées comme des tables) suivantes :

- interrogation (SELECT) ;
- insertion (INSERT) ;
- mise à jour (UPDATE) ;
- destruction (DELETE) ;
- référence à une table (REFERENCE).

Les bénéficiaires des droits sont les utilisateurs, les groupes ou les rôles. Les droits sont accordés par l'administrateur du SGBD. Cependant, il est possible de transférer la faculté de redistribuer ces droits à un autre utilisateur avec l'option WITH GRANT OPTION. L'instruction GRANT permet de réaliser l'association des droits et du bénéficiaire. Sa forme générale est la suivante :

```
GRANT <type de droit>
ON <objet>
TO <nom utilisateur>
```

Voici un exemple d'utilisation de GRANT.

L'administrateur crée un utilisateur « pastorius » avec l'identifiant « fender ».

```
CREATE USER pastorius IDENTIFIED BY fender ;
```

Il lui donne tous les droits sur la base de données 'pastorius_base' qu'il vient de créer. L'option 'ALL PRIVILEGES' sert à transmettre l'ensemble des droits dont dispose le donneur : en l'occurrence, l'administrateur dispose de tous les droits sur tous les objets. Le fait de préciser 'pastorius_base.*' signifie tous les sous-objets présents et à venir contenus dans la base de données 'pastorius_base'

```
CREATE DATABASE pastorius_base;
GRANT ALL PRIVILEGES ON pastorius_base.* TO 'pastorius' ;
```

L'utilisateur 'pastorius' crée la table 'jaco' dans la base de données 'pastorius_base' et donne l'accès de type « SELECT » à l'utilisateur 'nhop' préalablement créé.

```
USE pastorius_base;
CREATE TABLE jaco (NumMorceau INT PRIMARY KEY, Titre CHAR(50) ) ;
GRANT SELECT ON jaco TO 'nhop' ;
```

On obtient un message d'erreur : l'utilisateur 'pastorius' n'a pas le droit de retransmettre ses droits à un autre utilisateur. L'administrateur aurait dû entrer la commande suivante :

```
GRANT ALL PRIVILEGES ON pastorius_base.* TO pastorius WITH GRANT OPTION ;
```

Cette fois, la commande de transmission de droits peut être faite par l'utilisateur 'pastorius'

```
GRANT SELECT ON jaco TO 'nhop' ;
```

L'utilisateur 'nhop' a le droit d'effectuer toutes les opérations de consultation de la table 'jaco' de la base de données 'pastorius_base', il peut par exemple exécuter la séquence suivante :

```
USE pastorius_base;
SELECT * FROM jaco;
```

Lorsque l'on veut donner un droit à tous les utilisateurs du SGBD, on utilise le mot clé PUBLIC comme nom d'utilisateur.

```
GRANT SELECT ON jaco TO PUBLIC;
```

L'ensemble des utilisateurs peut alors interroger la table 'jaco' de la base de donnée 'pastorius_base' .On peut affiner ces permissions en ne les accordant que pour certains champs de la table. On spécifie alors pour le type de droit le ou les champs sur lesquels ils s'appliquent.

```
GRANT UPDATE Adresse ON jaco TO 'nhop' ;
```

L'utilisateur 'nhop' a le droit de mettre à jour le champ 'Adresse' de la table 'jaco'. Ce droit peut lui avoir été accordé par l'utilisateur 'pastorius' ou par l'administrateur du système. Si l'on ne spécifie pas le champ comme on l'a fait précédemment, le SGBD considère que le droit concerne tous les champs de la table.

Droits associés aux rôles

L'instruction GRANT permet également de donner un rôle à un utilisateur ou à un ensemble d'utilisateurs et s'utilise alors de la manière suivante :

```
GRANT <rôle>
 TO <liste des noms d'utilisateur séparés par des virgules>
```

Voici un exemple d'utilisation de GRANT avec les rôles. On considère les rôles suivants sur la table 'jaco' :

- consultation : interrogation ;
- utilisation : mise à jour et insertion ;
- gestion : destruction.

On crée les rôles et on met à jour les droits nécessaires.

```
CREATE ROLE consultation;
CREATE ROLE utilisation;
CREATE ROLE gestion;
GRANT SELECT ON jaco TO consultation;
GRANT UPDATE, INSERT ON jaco TO utilisation;
GRANT DELETE ON jaco TO gestion;
```

On affecte les rôles aux utilisateurs.

```
GRANT consultation TO 'pastorius', 'nhop', 'miller' ;
GRANT utilisation TO 'pastorius', 'miller' ;
GRANT gestion TO 'pastorius';
```

Les utilisateurs qui ont plusieurs rôles disposent des droits de tous les rôles auxquels ils appartiennent. Ainsi, l'utilisateur 'miller' possède les droits suivants sur la table : « INSERT, UPDATE, SELECT ». Pour retirer les droits accordés par l'instruction GRANT, on utilise l'instruction REVOKE**.** Elle est de la forme suivante :

```
REVOKE <type de droit>
ON <objet>
FROM <nom utilisateur>
```

Pour retirer les droits de l'utilisateur 'nhop' sur la table 'jaco', on procède comme suit :

```
REVOKE SELECT
ON jaco
FROM 'nhop' ;
```

Remarque

Seul l'utilisateur qui lui a donné ces droits peut les lui retirer. Dans notre cas, la commande précédente doit être lancée par l'utilisateur 'pastorius'.

Les droits donnés par plusieurs utilisateurs sont cumulatifs. Cela signifie que tous les droits accordés doivent être retirés pour qu'un utilisateur ne puisse plus effectuer les opérations. Ce « graphe » de permissions n'est pas toujours simple à gérer et devient rapidement de taille exponentielle. Par conséquent, les administrateurs des SGBD ont tendance à accorder parcimonieusement les possibilités de redistribuer des droits.

Pour aider l'administrateur, un bon SGBD procure une option de l'instruction de destruction d'un utilisateur, capable de détruire également tous les objets qu'il a créés. L'instruction DROP USER, non normalisée par SQL comme on l'a vu précédemment, s'utilise avec l'option CASCADE pour détruire les tables, les vues et autres que l'utilisateur a créés.

2.3 Utilisation des vues

La gestion des droits dans le SGBD, présentés à la section précédente, peut constituer un véritable casse-tête lorsque le nombre d'utilisateurs et d'objets augmente. En effet, les droits sur les tables et surtout sur les champs peuvent être définis par l'instruction GRANT, mais il devient vite fastidieux et particulièrement difficile de les modifier. Un des aspects de cette difficulté est le manque de lisibilité de l'ensemble des permissions.

La notion de rôle permet de « factoriser », ou rassembler, un ensemble de droits sur un ou plusieurs objets. Une approche complémentaire consiste à définir plus finement les objets sur lesquels on affecte les droits en se servant simplement des **vues SQL**. Celles-ci permettent de « cacher » certaines données à l'aide de critères de sélection précis, ce qui est impossible à réaliser d'une autre manière. De plus, les vues permettent de masquer aux utilisateurs la complexité de la structure des données, leur fournissant ainsi une interface plus commode avec la base de données.

Le mécanisme de création des vues a été abordé rapidement au chapitre 4. Les données ne sont pas stockées et sont recalculées à chaque utilisation de la vue. Une vue est le résultat d'une instruction SQL SELECT… : les possibilités de définition sont donc très étendues puisqu'il s'agit du cœur même du fonctionnement de SQL. Enfin, il est également possible de mettre à jour les données contenues dans les vues sous certaines conditions. Les permissions sur les vues sont ensuite accordées par les instructions de type GRANT utilisées précédemment. Voici quelques exemples de vues qui recourent à la base de données exemple 'casse'.

Vue utilisant une jointure simple

Le service marketing veut vérifier s'il n'y a pas de corrélations entre la couleur des véhicules vendus et la ville dans laquelle résident les clients qui les achètent. Il est inutile de fournir à ce service la procédure pour obtenir ces informations qui nécessite de lier les trois tables 'voiture', 'vente' et 'client'. On crée une vue qui contient uniquement la projection de ces deux informations et qui sélectionne les voitures vendues.

```
CREATE VIEW couleur_ville (Couleur, Ville)
AS SELECT voiture.couleur, client.ville
FROM voiture JOIN vente JOIN client ON voiture.NumVoit=vente.Numvoit
AND client.NumAch = vente.Numach ;
```

On donne les droits de visualiser ces informations aux personnes du service marketing qui sont les utilisateurs 'nhop' et 'miller'.

```
GRANT SELECT ON couleur_ville TO 'nhop', 'miller' ;
```

Vue utilisant une jointure plus élaborée

On souhaite que tous les utilisateurs puissent consulter le catalogue des voitures non encore vendues. Cette liste de voitures est accessible par une requête plus complexe que la précédente, de type jointure externe. Il s'agit d'un cas d'utilisation d'une vue, très pratique pour les utilisateurs qui n'ont pas à connaître les notions avancées de jointure externe.

```
CREATE VIEW catalogue (NumVoit, Marque, Type, Couleur)
AS SELECT voiture.NumVoit, voiture.Marque, voiture.Type, voiture.Couleur
FROM voiture LEFT OUTER JOIN vente ON voiture.NumVoit=vente.Numvoit
WHERE vente.Numvoit IS NULL;
```

On donne la permission à l'utilisateur PUBLIC, c'est-à-dire à tous les utilisateurs.

```
GRANT SELECT ON catalogue TO PUBLIC;
```

Vue avec possibilité de mise à jour

Le service comptabilité doit pouvoir accéder aux informations de vente et de clientèle afin de facturer et calculer le chiffre d'affaires. En outre, ce service a besoin de mettre à jour le prix de vente.

```
CREATE VIEW journal_compta (Date_de_vente, Prix_de_vente, Nom_client, Ville_client)
AS SELECT vente.DateVent, vente.Prix, client.Nom, client.Ville
FROM vente JOIN client ON vente.NumAch=client.NumAch
WITH CHECK OPTION ;
```

L'option CHECK OPTION permet les mises à jour des données au travers de la vue.

On donne les droits à un rôle que les utilisateurs 'pastorius' et 'miller' vont utiliser.

```
CREATE ROLE comptabilite ;
GRANT SELECT ON journal_compta TO comptabilite;
GRANT UPDATE Prix_de_vente ON journal_compta TO comptabilite;
```

On donne le rôle 'comptabilité' aux utilisateurs 'pastorius' et 'miller'.

```
GRANT comptabilite TO 'pastorius', 'miller'
```

L'utilisation des vues permet de définir des objets dynamiques puisqu'une vue est le résultat d'une requête et n'est jamais stockée. Les vues s'utilisent en complément du système général de droits présenté à la section précédente. Il s'agit de la bonne solution pour éviter la complexité de description des permissions sur la totalité des champs. Comme pour le reste de la norme SQL, tous les SGBD ne proposent pas les vues.

3 Transactions

Cette section présente plusieurs outils procurés par les SGBD pour protéger les opérations effectuées sur les données. Les incidents liés aux données dans un SGBD proviennent essentiellement de l'**accès concurrent** à ces dernières par plusieurs utilisateurs. Ces problèmes sont habituellement résolus par les mécanismes associés aux **transactions** présentées dans cette section. Celles-ci permettent également de résoudre les pertes dues aux

erreurs de manipulation ou celles liées aux erreurs dans le traitement des données, par exemple en cas de panne matérielle.

3.1 Accès concurrent aux données

La section précédente aborde la définition des accès et des permissions sur les différents objets gérés par le SGBD. Implicitement, cela signifie que plusieurs utilisateurs ou applications peuvent accéder aux données en même temps. De surcroît, la plupart des machines sont connectées à un réseau, ce qui augmente encore le nombre potentiel d'utilisateurs simultanés.

Lecture(s) étrange(s)

L'accès multiple en lecture ne pose pas habituellement de problèmes, mais que se passe-t-il lorsqu'un ou plusieurs utilisateurs décident de modifier les mêmes données au même moment ? On considère la séquence d'instructions suivante appliquée à la base de données « casse ». On suppose que tous les utilisateurs disposent de tous les droits sur tous les objets (tables et champs) de cette base de données :

- L'utilisateur 'pastorius' consulte la liste des voitures non vendues.
- L'utilisateur 'nhop' enregistre la vente d'une voiture.
- L'utilisateur 'pastorius' relance la même requête et trouve un résultat différent.
- L'utilisateur 'nhop' invalide la vente de cette voiture.
- L'utilisateur 'pastorius' pensant s'être trompé relance la même requête qui aboutit au résultat initial.

Pour trois exécutions de la même requête, l'utilisateur 'pastorius' va obtenir des résultats différents. On peut considérer la même séquence exécutée dans un ordre différent.

- L'utilisateur 'pastorius' consulte la liste des voitures non vendues.
- L'utilisateur 'pastorius' relance la requête et trouve le même résultat.
- L'utilisateur 'nhop' enregistre la vente d'une voiture.
- L'utilisateur 'nhop' invalide la vente de cette voiture.
- L'utilisateur 'pastorius' relance la requête précédente et retrouve le même résultat.

Comment décider de l'ordre dans lequel exécuter les instructions ? En pratique, le SGBD ne dispose pas d'éléments de comparaison qui lui permettraient d'ordonner la série d'instructions afin que cela passe inaperçu comme dans la deuxième série. On pourrait utiliser ici le terme de **lecture fantôme** pour qualifier les problèmes rencontrés : une donnée apparaît puis disparaît.

Incohérence de résultats

On peut imaginer une autre série d'instructions qui réalisent des modifications de données effectuées par différents utilisateurs. En raison de l'augmentation des frais de structure, le service comptable a décidé d'une augmentation générale du prix de vente de 5 %. Afin que cette dernière passe inaperçue et dans le cadre d'une campagne de communication, le service marketing offre 100 euros de ristourne sur tout le catalogue pendant un mois. L'utilisateur 'pastorius' appartient au service comptable et l'utilisateur 'nhop' au service marketing. Les vérifications sont effectuées par l'utilisateur 'miller' de la direction (voir figure 6.2).

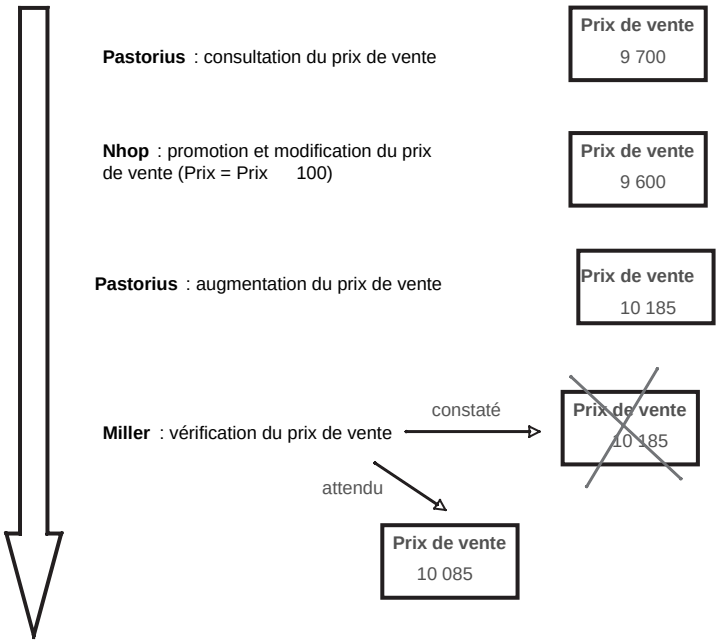
- L'utilisateur 'pastorius' consulte le prix de vente de la voiture 'X'.

- L'utilisateur 'nhop' effectue la modification de promotion sans vérifier le prix de vente (Prix_vente = Prix_vente – 100).
- L'utilisateur 'pastorius' effectue la modification d'augmentation (Prix_vente = Prix_vente × 1,05).

L'utilisateur 'miller' vérifie le résultat de l'opération de modification du prix de vente et constate une différence avec le résultat attendu : le prix est égal à (Prix_vente – 100) × 1,05, alors qu'il s'attendait à un prix égal à (Prix_vente × 1,05) – 100.

Figure 6.2

Déroulement de la séquence d'instructions de mise à jour du prix de vente.



On obtient alors une « incohérence » due à la séquence de modification des données. Les opérations effectuées sur le champ 'Prix' ne sont en effet pas commutatives. Là encore, le SGBD ne dispose pas d'éléments lui permettant d'ordonner correctement les opérations. En revanche, pour certains prix de vente, la mise à jour pourra être correcte.

Il existe bien d'autres types d'incohérences qui peuvent être provoquées par l'accès concurrent aux données. On peut citer le cas des **anomalies de lecture** : « Deux lectures successives ne donnent pas le même résultat. » Cela survient lorsque la modification d'une donnée est effectuée par un processus concurrent entre deux lectures successives. Le SGBD traite naturellement les requêtes de manière séquentielle, dans l'ordre de leur arrivée. Si l'on augmente le nombre d'utilisateurs, ce type de problèmes peut évidemment se multiplier.

Verrous

On a donc besoin de disposer d'un mécanisme qui garantisse une certaine « exclusivité » sur les données lors des opérations de mise à jour. Un tel mécanisme existe depuis longtemps en informatique pour résoudre ce problème : il s'agit des **verrous**. L'idée est simple : on bloque une ressource pour effectuer les opérations et on la libère dès que les opérations sont effectuées. Cette méthode semble résoudre le problème ; elle présente cependant quelques pièges et doit être utilisée avec prudence. En effet, on peut rapidement parvenir à une situation de blocage que l'on nomme **étreinte fatale** (*deadlock* en anglais) ou parfois

interblocage. Pour illustrer cette situation, on suppose que deux vendeurs veulent mettre à jour la base de données 'casse' :

- L'utilisateur 'pastorius' veut insérer une vente de voiture dans la table 'vente' et constate à cette occasion une erreur sur la couleur dans la table 'voiture' qu'il veut modifier.
- L'utilisateur 'miller' veut insérer à la fois une nouvelle voiture dans la table 'voiture' et la mention de sa vente dans la table 'vente'.

La séquence d'instruction peut être la suivante :

- L'utilisateur 'pastorius' pose un verrou sur la table 'vente'.
- L'utilisateur 'miller' pose un verrou sur la table 'voiture'.
- L'utilisateur 'pastorius' a terminé la mise à jour de 'vente' et veut réaliser celle de 'voiture'… mais la table 'voiture' est verrouillée par 'miller'.
- L'utilisateur 'miller' a terminé la mise à jour de 'voiture' et veut procéder à celle de 'vente'… mais la table 'vente' est verrouillée par 'pastorius.

On parvient à une situation où les deux utilisateurs attendent que l'autre libère la ressource pour continuer. Évidemment, c'est une attente infinie qui se profile pour chacun. D'autres phénomènes de blocage peuvent survenir dans l'utilisation des verrous. On parle parfois dans ce cas de **famine**. Voici un exemple illustrant cette possibilité.

L'utilisation classique des verrous pour protéger l'accès à une ressource est la suivante (algorithme des « lecteurs-rédacteurs ») :

- Tous les lecteurs peuvent lire en même temps.
- Si un rédacteur veut écrire, aucun lecteur ne doit plus utiliser la donnée.
- Si un rédacteur est en train d'écrire, aucun lecteur ne peut lire la donnée et aucun autre rédacteur ne peut y accéder (accès exclusif).

S'il existe un grand nombre de lecteurs qui se succèdent en lecture sur la donnée, l'attente pour un rédacteur qui voudrait la modifier peut alors devenir quasi infinie.

Laisser la gestion des verrous à un utilisateur est donc délicat et peut conduire à des blocages du système. Les SGBD performants disposent d'outils capables de détecter et résoudre ces phénomènes, voire de les prévenir le cas échéant. Les algorithmes utilisés à cet effet dépassent le cadre de cet ouvrage.

3.2 Mécanisme des transactions

Pour résoudre les problèmes d'accès concurrents vus précédemment, les SGBD proposent aux utilisateurs un outil que l'on nomme une **transaction**. L'idée est de considérer un ensemble d'instructions comme une seule instruction. L'ensemble d'instructions est dit **unitaire** ou **atomique**.

Propriétés des transactions

La notion d'atomicité peut être décrite en ces termes :

- Soit le SGBD est capable d'exécuter toutes les instructions qui composent une transaction et il effectue les mises à jour provoquées par ces instructions.
- Soit il n'y parvient pas et il remet la base de données dans l'état cohérent précédent le début de l'exécution des instructions de la transaction.

Les propriétés que doivent vérifier les transactions sont résumées par le terme **ACID :**

- **Atomicité.** Une transaction est atomique : elle est exécutée entièrement ou abandonnée.
- **Cohérence.** La transaction doit se faire d'un état cohérent de la base vers un autre état cohérent.
- **Isolement.** Des transactions simultanées ne doivent pas interférer entre elles.
- **Durabilité.** La transaction a des effets permanents même en cas de panne.

Les qualités des transactions sont un argument de vente pour un SGBD. Il est en effet complexe de trouver un compromis entre la sécurité et les performances. Bien qu'il s'agisse d'un mécanisme éprouvé qui existe depuis fort longtemps, les transactions ainsi que les techniques de gestion des accès concurrentiels restent des sujets de recherches importants dans les laboratoires. Les transactions font partie de la norme SQL.

Remarque

Les transactions permettent de résoudre les problèmes liés aux accès concurrentiels, mais également de traiter le cas de la reprise en cas de panne. Le SGBD est ainsi capable de remettre la base de données dans l'état cohérent où elle se trouvait avant le début de la transaction qui a échoué pour cause de panne. Les transactions sont également employées pour annuler des erreurs de traitement éventuelles. En effet, grâce à ce mécanisme, on peut revenir à l'état initial dans lequel se trouvait la base de données avant le début de la série de mauvaise(s) manipulation(s) sur la base de données. Il s'agit certainement de l'utilisation principale des transactions.

Réalisation des transactions dans les SGBD

Pour mettre en œuvre les transactions, le SGBD doit offrir **l'exclusivité d'accès** aux données ainsi que la possibilité **d'annuler** des modifications effectuées. Afin de garantir l'accès exclusif aux données, le SGBD utilise les verrous vus précédemment. Ces mécanismes sont associés à des algorithmes de protection sophistiqués afin de prévenir les problèmes des méthodes de verrouillage.

La possibilité de revenir en arrière repose sur l'utilisation d'un journal de transactions. Pour ce faire, le SGBD garde une trace de toutes les requêtes de la transaction et des données qui sont modifiées par ces instructions. Ce mécanisme est identique à celui qui est utilisé pour les systèmes de fichiers modernes dans les systèmes d'exploitation. Puisque la plupart des écritures se font en mémoire pour gagner du temps, le système de fichiers n'est pas forcément cohérent en cas de panne de courant qui entraîne la perte de toutes les informations se trouvant en mémoire. On conserve donc un journal de l'ensemble des lectures et écritures afin de pouvoir reconstituer un système cohérent. En utilisant ce journal, à partir d'un état courant du système, on est capable de réexécuter les instructions qui n'ont pu l'être effectivement ou on revient en arrière si ce n'est pas possible.

La description faite ci-dessus est évidemment simpliste par rapport à la réalité plus complexe des SGBD. Les méthodes de verrouillage implémentées dans ces derniers sont bien plus élaborées que celles présentées ici. Il en est de même pour les mécanismes de journalisation qui disposent de plusieurs niveaux.

Différents niveaux de transaction

La norme SQL fixe plusieurs degrés de qualité pour les transactions : on parle de niveaux **d'isolation**. C'est-à-dire qu'une transaction pourra être rendue plus ou moins perméable aux autres transactions. Les niveaux standard sont les suivants :

- 0, READ UNCOMMITED. Lecture de données non validées en cours de transaction (niveau le plus bas).
- 1, READ COMMITED. Modification des données possible en cours de transaction.
- 2, REPEATABLE READ. Insertion de nouveaux enregistrements possible en cours de transaction.
- 3, SERIALIZABLE. Toutes les transactions sont traitées en « série » (niveau le plus sûr).

Pourquoi prévoir plusieurs niveaux, puisque le but est d'obtenir une sécurité maximale ? La sécurité a un coût en matière de performances et, dans certains contextes, il n'est pas nécessaire de demander au SGBD de traiter une requête avec le niveau maximal d'exclusivité. Il est clair que, pour une simple opération de lecture, le niveau 1 qui garantit l'impossibilité de lire des données non validées peut être suffisant.

Remarque

Les SGBD ne fonctionnent pas par défaut au même niveau d'isolation. Il est prudent de vérifier dans la documentation à quel niveau se trouve par défaut le SGBD que l'on utilise. En effet, certains SGBD très répandus fonctionnent par défaut au niveau 1 ! Ce peut être suffisant pour certaines applications, encore faut-il en être bien conscient. Le choix du niveau est motivé par des arguments plus commerciaux que techniques : le SGBD est plus rapide en utilisant un niveau moins élevé.

Syntaxe SQL des transactions

L'instruction pour débuter une transaction est START TRANSACTION. La transaction prendra fin par l'une des instructions suivantes :

- COMMIT. Les instructions effectuées depuis START TRANSACTION sont validées.
- ROLLBACK. Les instructions effectuées depuis START TRANSACTION sont annulées.

Voici un exemple d'insertion et de mise à jour dans la base de données 'casse' que l'on annule ensuite :

```
#Démarrage de la transaction
START TRANSACTION ;
#Insertion d'un tuple dans la table 'client'
INSERT INTO client VALUES (6,'Laetitia',34,'Paris','F');
#Mise à jour des données de prix (augmentation de 5 %)
UPDATE vente SET Prix=Prix*1.05 ;
#Annulation des modifications faites depuis START TRANSACTION
ROLLBACK ;
```

Voici un exemple de destruction dans la table 'client' de la base de données 'casse' que l'on valide ensuite :

```
#Démarrage de la transaction
START TRANSACTION ;
#Destruction des tuples dont le champ 'Ville' vaut 'Paris' dans la table 'client'
DELETE FROM client WHERE Ville='Paris';
#Validation des modifications faites depuis START TRANSACTION
COMMIT ;
```

On peut choisir le niveau d'isolation au moment où l'on démarre la transaction. Dans l'exemple suivant, on fixe le niveau maximal de sécurité.

```
START TRANSACTION
ISOLATION LEVEL SERIALIZABLE ;
…
```

« Points de retour » dans les transactions

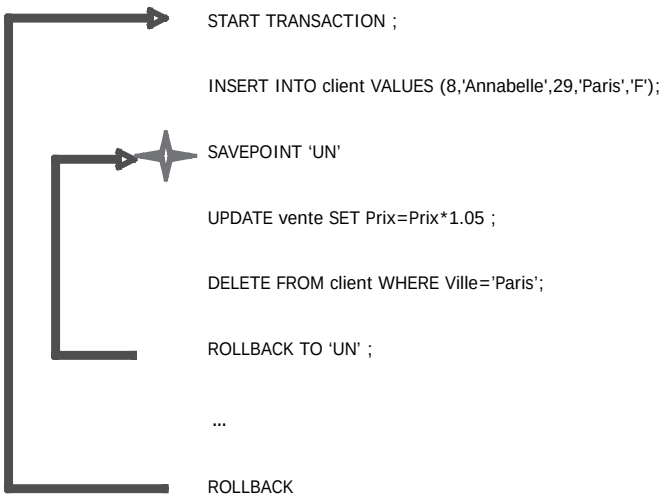
Les instructions contenues dans une transaction sont considérées par le SGBD comme « unitaires », c'est-à-dire qu'elles sont exécutées d'un seul bloc ou ne sont pas exécutées. L'instruction ROLLBACK annule toutes les instructions effectuées depuis l'instruction START TRANSACTION. Cependant, il est possible de diviser cet ensemble d'instructions en sous-ensembles par la définition de « points de retour » utilisés par l'instruction ROLLBACK. Au lieu d'annuler les instructions exécutées depuis START TRANSACTION, on revient à l'état de la base de données depuis le point défini par l'instruction SAVEPOINT (voir figure 6.3).

```
#Démarrage de la transaction
START TRANSACTION ;
#Insertion d'un tuple dans la table 'client'
INSERT INTO client VALUES (8,'Annabelle',29,'Paris','F');
# Définition d'un point de retour que l'on nomme 'UN'
SAVEPOINT 'UN'
#Mise à jour des données de prix (augmentation de 5 %)
UPDATE vente SET Prix=Prix*1.05 ;
#Destruction des tuples dont le champ 'Ville' vaut 'Paris' dans la table 'client'
DELETE FROM client WHERE Ville='Paris';
#Annulation des modifications faites depuis le point 'UN'
ROLLBACK TO 'UN' ;
```

Une différence essentielle par rapport à l'instruction ROLLBACK vue précédemment est que, dans ce cas, la transaction n'est pas terminée. Un autre ROLLBACK reviendrait à l'état de la base de données avant l'exécution de l'instruction START TRANSACTION.

Figure 6.3

Point de retour dans une transaction.



Le mécanisme des points de retour est très utile pour prévenir les cas de mauvaise manipulation de données. On combine ainsi la possibilité d'effectuer les modifications dans une seule transaction avec celle de ne pas annuler toutes les modifications. Certaines d'entre elles prennent en effet un temps considérable.

Remarque

La plupart des SGBD considèrent que chaque instruction SQL est en soi une transaction qui est automatiquement validée (mode AUTO COMMIT). Certains fonctionnent en créant une nouvelle transaction lors de chaque connexion au SGBD, ce qui permet d'annuler le cas échéant toutes les opérations effectuées depuis la connexion. Dans le doute, il est donc préférable de spécifier explicitement le début de toute transaction par une instruction START TRANSACTION.

Le système des transactions gérées par le SGBD est un outil indispensable pour prendre en compte les problèmes cités précédemment : la concurrence d'accès, les erreurs et les reprises après les pannes. Comme on l'a vu dans cette section, la réalisation des transactions fait appel à des compromis entre la rapidité et la sécurité. Tous les SGBD n'effectuent pas les mêmes choix et présentent des degrés différents de fiabilité. Les cas d'interblocage peuvent être fréquents dans certains SGBD pourtant très utilisés. D'autres, également fort répandus, ne proposent même pas les transactions sur leurs tables en « standard ».

4 Triggers

Les **triggers** ou déclencheurs ont un objectif différent des outils tels que les transactions. Ils servent à exécuter des contrôles complémentaires personnalisés au moment des opérations d'ajout, de suppression et mise à jour des données. Ils sont utiles également pour effectuer un formatage spécifique des données. Pratiquement, un trigger est un ensemble d'instructions SQL, définies par l'utilisateur, qui seront déclenchées lors d'une action d'ajout, de suppression ou de mise à jour de données. Il s'applique donc à un objet de type table. On peut choisir d'exécuter un trigger avant ou après une instruction de mise à jour. Les données manipulées par le trigger sont stockées dans des tables temporaires que l'on désigne dans le code du trigger par les termes :

- NEW valeurs, après l'exécution de l'opération de mise à jour ;
- OLD valeurs, avant l'exécution de l'opération de mise à jour.

4.1 Syntaxe générale pour créer et détruire un trigger

L'instruction, normalisée par la norme SQL, est de la forme générale suivante :

```
CREATE TRIGGER 'Nom du trigger'
'Moment où le trigger est exécuté'
'Opération concernée'
ON 'Nom de la table'
FOR EACH ROW(ou STATEMENT)
BEGIN
'Instructions
END
```

- Le moment peut prendre les valeurs BEFORE ou AFTER.
- Les opérations concernées sont INSERT, UPDATE et DELETE.
- La différence entre ROW et STATEMENT est que l'on exécute les instructions pour chaque ligne ou pour toute la table.

Pour détruire un trigger, on utilise l'instruction DROP et on spécifie le nom du trigger que l'on veut détruire.

```
DROP TRIGGER conversion
```

Comme toujours, la syntaxe peut être légèrement différente suivant le SGBD employé.

Contrôle de la forme du contenu des champs

Pour le formatage, on veut imposer que les noms d'une ville dans la table 'client' soient transformés en majuscules avant d'être insérés. On fait référence ici à la nouvelle table 'NEW' que l'on modifie avant l'insertion.

```
CREATE TRIGGER ville_majuscule BEFORE INSERT ON client FOR EACH ROW
BEGIN
SET NEW.Ville=UPPER(NEW.Ville) ;
END
```

Modification automatique du contenu d'un champ

Les triggers permettent, par exemple, de compléter le mécanisme d'intégrité référentielle par des contrôles plus fins et la définition des actions associées à ces derniers. On considère l'exemple de la base de données 'casse'. Si le prix de vente est supérieur à 40 000 lors d'une insertion de données dans la table 'vente', on suppose que la personne a saisi par erreur le prix en francs et non pas en euros. On effectue automatiquement la conversion des francs en euros. Là encore, on modifie les données de la table 'NEW' qui contient celles que l'on va insérer.

```
CREATE TRIGGER conversion BEFORE INSERT ON vente FOR EACH ROW
BEGIN
IF New.Prix> 40000
THEN SET
NEW.Prix=NEW.Prix/6.5596 ;
END IF;
END
```

Mise à jour d'autres tables

On peut également provoquer la mise à jour d'autres tables à l'aide d'un trigger. Si l'on ajoute un champ 'compte' à la table 'client' qui représente l'état du solde de son compte vis-à-vis de l'entreprise, on veut pouvoir mettre à jour automatiquement le champ 'compte' de l'acheteur lors de l'insertion d'une opération de vente. À cette occasion, on soustrait le prix de vente de la voiture du montant du compte du client. C'est la seule manière de réaliser cette opération de modification d'une autre table lors d'une simple insertion.

```
# Ajout d'un champ 'compte' à la table 'client'
ALTER TABLE client ADD COLUMN compte INT ;
# Création du trigger
CREATE TRIGGER compte BEFORE INSERT ON vente FOR EACH ROW
BEGIN
UPDATE client
SET client.Compte=client.Compte-NEW.Prix
WHERE client.NumAch=New.NumAch ;
END
# Ajout d'une vente dans la table 'vente'
INSERT INTO vente VALUES (6, '2005-03-01',50000,2,5)
```

Le compte du client de numéro 'NumAch' égal à '2' sera automatiquement diminué de la somme de '50000'.

Un trigger apporte de la souplesse et de la personnalisation dans la définition des contraintes que l'on associe aux tables. Pour ce faire, il déclenche l'exécution d'un morceau de code spécifique associé à un événement de mise à jour de la table. Il est parfois possible de réaliser les opérations d'une autre de manière plus complexe. Du point de vue de la sécurité des données, le code associé au trigger s'exécute sur le serveur sans interaction avec le client. On réduit ainsi les risques d'erreur liés aux échanges entre le client et le serveur.

Résumé

Ce chapitre expose les différentes dispositions que l'on doit prendre afin de prévenir l'altération ou la perte des données. On a présenté tout d'abord les précautions de premier niveau qu'il ne faut pas négliger. Elles concernent essentiellement l'environnement dans lequel se trouve le SGBD : la machine munie de son système d'exploitation, les locaux informatiques et enfin le réseau sur lequel elle est connectée. Un autre point important concerne la sauvegarde et la duplication des données, de façon à garantir l'accessibilité à ces dernières même en cas de sinistre.

Une fois l'environnement du SGBD sécurisé, on a abordé les permissions et les restrictions qui sont gérées directement par le SGBD. Il s'agit de la partie prise en charge par l'administrateur du SGBD qui définit les droits sur les différents objets de la base de données. Cette gestion devient rapidement complexe : on utilise en complément les vues SQL pour décrire plus finement les objets sur lesquels on distribue les droits.

Les précautions précédentes mises en œuvre, on se trouve confronté au problème de l'accès concurrent aux données. La solution consiste en l'utilisation de verrous. Leur mécanisme est très délicat à gérer et l'on préfère laisser ce soin au SGBD. Ce dernier procure de surcroît un mécanisme de « journalisation » des instructions capable de remettre la base de données dans l'état cohérent précédant la séquence d'instructions. Cette combinaison des mécanismes d'accès exclusif associés à la journalisation se nomme les *transactions*. Enfin, les *triggers* (ou déclencheurs en français) sont très utiles pour compléter la panoplie d'outils qui assurent le contrôle des données d'une base de données.

Exercices

EXERCICE 1 SÉCURITÉ DE BASE

Énoncé

Quelles sont les vérifications de base indispensables avant de faire héberger un SGBD serveur de bases de données par une structure ?

Solution

La décision doit reposer sur la prise en considération de plusieurs points ; la liste ci-dessous n'est pas exhaustive.

Aspects techniques. L'une des premières préoccupations doit être d'auditer le type de machine ainsi que le système d'exploitation sur lequel sera hébergé le SGBD. D'autre part, les SGBD sont gourmands en ressources de calcul et de stockage et il faut donc s'assurer que l'on disposera du nécessaire dans ces domaines.

On peut supposer que la machine sera connectée à un réseau qu'il faut auditer également. De quel type de connexion dispose l'hébergeur, quels matériels de protection contre les intrusions utilise-t-il ? Le minimum est de disposer d'un dispositif de filtrage pour se protéger des intrusions liées au réseau. Un plus est de disposer de deux fournisseurs d'accès, l'un prenant le relais en cas de faille de l'autre.

Enfin, quel matériel de sauvegarde peut-on utiliser, et de quelle taille de stockage dispose-t-on ? Une sauvegarde quotidienne est indispensable pour une base de données standard, c'est-à-dire sur laquelle les mises à jours sont raisonnablement fréquentes. En cas de mises à jour intensives, on peut prévoir de conserver plusieurs versions de la base sauvegardée.

Aspects humains. Les machines et les réseaux aussi performants soient-ils sont gérés avant tout par des êtres humains. Il est donc essentiel de savoir combien de personnes assurent cette mission, et de quelle façon ; d'identifier la politique de sécurité générale : accès physique aux machines, protection contre les intrusions réseau et le suivi des incidents, mise à jour des machines, gestion des comptes …

EXERCICE 2 DISPONIBILITÉ DES DONNÉES

Énoncé

Quel type d'architecture permet de garantir en même temps la haute disponibilité des données et de disposer de sauvegarde ? Quels sont les inconvénients de ce système ?

Solution

La haute disponibilité nécessite que l'accès aux données soit d'une part permanent et d'autre part performant. Afin de garantir que les données soient disponibles en permanence, on doit effectuer des copies à intervalles réguliers des données qui proviennent d'une base de données « primaire » vers des bases de sauvegarde « secondaires ». En cas de panne du primaire, on peut basculer rapidement vers l'un des secondaires. L'idéal est de disposer de plusieurs secondaires dans des lieux géographiquement éloignés, situés sur des réseaux (informatiques) différents.

Disposer en outre d'un accès performant suppose que l'on utilise également les secondaires pour faire de la répartition de charge. Le système de répartition choisi doit tenir

compte de la « proximité », au sens informatique, du serveur par rapport au client qui effectue la requête. Cette « proximité » peut être calculée en termes de distance – combien de nœuds du réseau sont traversés ? – mais également en termes de charge des branches du réseau. Il existe des outils capables de tenir à jour en permanence ce genre d'informations, mais ils sont assez complexes à gérer. Lors du choix de la machine vers laquelle on va rediriger la demande, on doit prendre en considération également sa charge et donc son aptitude à répondre rapidement.

Les inconvénients d'une solution complète de ce type sont évidents. Les applications de répartition de charge sur un réseau standard sont déjà complexes à gérer ; on imagine bien qu'elles le sont d'autant plus lorsque l'on passe à une échelle supérieure. Un autre inconvénient est la difficulté de maintenir des copies des données à jour par rapport à la base primaire. Si les mises à jour sont très fréquentes, il devient difficile de disposer des mêmes données sur tous les secondaires.

Une autre solution plus facile à gérer pour répartir « naturellement » la charge entre les serveurs consiste à répartir les données entre ces serveurs. Ainsi, cette opération se fait par l'architecture même de la base de données qui est dite « répartie ». En revanche, cette solution ne résout pas le problème de la recopie des données. Si l'un des serveurs tombe en panne, il n'existe pas, contrairement au système précédent, de mécanisme automatique pour le remplacer. On peut envisager des variantes qui panachent les deux systèmes : par exemple, on peut mettre en place un serveur maître qui recopie une partie des données sur les secondaires. Cette problématique générale est illustrée par les grands moteurs de recherche, ou encore les serveurs de vidéo qui utilisent plusieurs milliers de serveurs répartis sur le réseau Internet.

EXERCICE 3 NIVEAUX D'UTILISATION D'UNE BASE DE DONNÉES

Énoncé

Lors de la conception d'une base de données, on doit envisager dès le début les différentes catégories d'utilisateurs qui vont l'utiliser. On considère l'exemple complet « livraison de pizzas » traité au chapitre 5, « Du langage parlé à SQL ». Quels niveaux d'utilisation des données doit-on prévoir ? On se limite aux grandes catégories.

Solution

La notion d'utilisateur ne représente pas forcément une personne physique ; il s'agit le plus souvent d'applications qui disposent ou non de droits sur certaines données. En première approche, on peut distinguer trois grandes catégories d'utilisateurs de la base de données.

- Les clients peuvent consulter les informations les concernant ainsi que le catalogue des pizzas disponibles avec leur prix et leur composition.
- L'activité courante, qui par conséquent va générer les mises à jour les plus fréquentes, est la commande de pizzas. Les utilisateurs concernés sont les employés qui mettent à jour les données de la table 'commande', mais également celles de la table 'client' pour créer de nouveaux clients le cas échéant. Ils ont accès en lecture à toutes les données des autres tables.
- Enfin, les gestionnaires de l'activité doivent pouvoir mettre à jour toutes les informations de gestion : en particulier, celles sur les pizzas, les clients, les livreurs et les voitures utilisées. En revanche, on peut considérer qu'ils n'ont pas besoin de modifier les informations concernant les commandes, mais qu'ils doivent pouvoir y accéder en lecture.

On résume dans un tableau (voir tableau 6.1) les droits par catégories sur les tables (par « Écriture », on entend mise à jour, insertion et destruction).

Figure 6.1 : Synthèse des droits par catégories de la base de données « pizza »

	commande	pizza	compose	ingredient	tarification	client	livreur	vehicule
Client	X	Lecture	Lecture	Lecture	Lecture	Lecture	X	X
Employé	Lecture/ Écriture	Lecture	Lecture	Lecture	Lecture	Lecture	Lecture	Lecture
Gestion-naire	Lecture	Lecture/ Écriture	Lecture/ Écriture	Lecture/ Écriture	Lecture/ Écriture	Lecture/ Écriture	Lecture/ Écriture	Lecture/ Écriture

EXERCICE 4 GESTION DES DROITS

Énoncé

Comment représenter pratiquement les catégories identifiées à l'exercice précédent dans un SGBD ? Écrivez les instructions permettant de le faire.

Solution

Si le SGBD le supporte, la solution la plus élégante est d'utiliser les rôles. On crée un rôle par catégorie et on lui affecte les permissions. Il suffira ensuite de donner ce rôle aux utilisateurs ou aux applications concernées. Dans cet exemple, on a surtout cherché à mettre en évidence tous les droits à distribuer. Il existe des notations plus concises.

Création du rôle 'client' et distribution des droits. `CREATE ROLE client;`

```
GRANT SELECT ON pizza TO client;
GRANT SELECT ON compose TO client;
GRANT SELECT ON ingrédient TO client;
GRANT SELECT ON tarification TO client ;
GRANT SELECT ON client TO client;
```

Création du rôle 'employe' et distribution des droits. `CREATE ROLE employe;`

```
GRANT SELECT ON pizza TO employe ;
GRANT SELECT ON compose TO employe ;
GRANT SELECT ON ingrédient TO employe ;
GRANT SELECT ON tarification TO employe ;
GRANT SELECT ON vehicule TO employe ;
GRANT SELECT ON livreur TO employe ;
GRANT SELECT ON client TO employe ;
GRANT UPDATE ON client TO employe;
GRANT DELETE ON client TO employe;
GRANT INSERT ON client TO employe;
GRANT SELECT ON commande TO employe ;
GRANT UPDATE ON commande TO employe;
GRANT DELETE ON commande TO employe;
GRANT INSERT ON commande TO employe;
```

Création du rôle 'gestionnaire' et distribution des droits. `CREATE ROLE gestionnaire;`

```
GRANT SELECT ON commande TO gestionnaire ;
GRANT SELECT ON client TO gestionnaire;
GRANT UPDATE ON client TO gestionnaire;
GRANT DELETE ON client TO gestionnaire;
GRANT INSERT ON client TO gestionnaire;
GRANT SELECT ON livreur TO gestionnaire;
GRANT UPDATE ON livreur TO gestionnaire;
```

```
GRANT DELETE ON livreur TO gestionnaire;
GRANT INSERT ON livreur TO gestionnaire;
GRANT SELECT ON vehicule TO gestionnaire;
GRANT UPDATE ON vehicule TO gestionnaire;
GRANT DELETE ON vehicule TO gestionnaire;
GRANT INSERT ON vehicule TO gestionnaire;
GRANT SELECT ON tarification TO gestionnaire;
GRANT UPDATE ON tarification TO gestionnaire;
GRANT DELETE ON tarification TO gestionnaire;
GRANT INSERT ON tarification TO gestionnaire;
GRANT SELECT ON ingrédient TO gestionnaire;
GRANT UPDATE ON ingrédient TO gestionnaire;
GRANT DELETE ON ingrédient TO gestionnaire;
GRANT INSERT ON ingrédient TO gestionnaire;
GRANT SELECT ON compose TO gestionnaire;
GRANT UPDATE ON compose TO gestionnaire;
GRANT DELETE ON compose TO gestionnaire;
GRANT INSERT ON compose TO gestionnaire;
GRANT SELECT ON pizza TO gestionnaire;
GRANT UPDATE ON pizza TO gestionnaire;
GRANT DELETE ON pizza TO gestionnaire;
GRANT INSERT ON pizza TO gestionnaire;
```

On distribue les rôles à des utilisateurs :

- 'swallow' et 'clarke' sont des clients.

```
GRANT client'TO 'swallow, 'clarke';
```

- 'gomez' et 'pattitucci' sont des employés.

```
GRANT employe TO 'gomez', 'pattitucci';
```

- 'pastorius' est un gestionnaire.

```
GRANT gestionnaire TO 'pastorius';
```

EXERCICE 5 VUES

Énoncé

De nouveaux utilisateurs aux besoins plus spécifiques vont utiliser la base de données de livraison de pizzas vue au chapitre 5, « Du langage parlé à SQL ».

- Le service comptabilité doit rémunérer les livreurs : il a besoin de connaître le nombre de commandes livrées par chacun d'eux.
- Le service du personnel effectue un suivi des livreurs : il a besoin de connaître le nombre de retards de chacun, mais également le nombre de commandes livrées.
- Un laboratoire de recherche en sociologie effectue des études sur la clientèle, spécifiquement sur la relation entre les ingrédients des pizzas achetées et l'adresse des clients.

Ces utilisateurs ne doivent accéder qu'aux champs dont ils ont strictement besoin.

Solution

On remarque que les informations dont ces utilisateurs ont besoin ne sont pas directement disponibles dans une table ni même dans un champ particulier. Il s'agit de résultats de calculs sur l'ensemble de la base de données. Le moyen le plus simple pour gérer ces besoins est de créer des vues pour chaque catégorie d'utilisateurs, auxquels on distribuera les permissions sur ces vues. Il est évidemment possible de créer des rôles d'utilisateurs comme dans l'exercice précédent, ce qui est plus élégant mais n'est pas supporté par tous les SGBD.

Service comptabilité. Le besoin est uniquement d'avoir accès aux nombres de commandes livrées pour chaque livreur. Il s'agit du résultat d'un calcul classique sur un agrégat. On affiche uniquement les informations du livreur (code et nom) et le résultat du calcul.

```
SELECT L.CodeLivreur, L.NomLivreur, COUNT(*) AS Nombre_Livraison
FROM commande C JOIN livreur L ON C.CodeLivreur=L.CodeLivreur
GROUP BY C.CodeLivreur
ORDER BY L.NomLivreur;
```

On crée une vue à partir de cette requête :

```
CREATE VIEW commande_livreur AS
SELECT L.CodeLivreur, L.NomLivreur, COUNT(*) AS Nombre_Livraison
FROM commande C JOIN livreur L ON C.CodeLivreur=L.CodeLivreur
GROUP BY C.CodeLivreur
ORDER BY L.NomLivreur;
```

On distribue les droits de lecture à l'utilisateur 'comptabilité'

```
GRANT SELECT ON commande_livreur TO comptabilite;
```

Service du personnel. Par un raisonnement identique, on obtiendrait la requête qui permet de calculer le nombre de retards cumulés par livreur.

```
CREATE VIEW retard_livreur AS
SELECT L.CodeLivreur, L.NomLivreur, COUNT(*) AS Nombre_Retard
FROM commande C JOIN livreur L ON C.CodeLivreur=L.CodeLivreur
WHERE C.Retard='O'
GROUP BY C.CodeLivreur
ORDER BY L.NomLivreur;
```

On distribue les droits de lecture à l'utilisateur 'personnel'.

```
GRANT SELECT ON retard_livreur TO personnel;
```

Le service du personnel a besoin également d'utiliser la précédente vue :

```
GRANT SELECT ON commande_livreur TO personnel;
```

Pour être complet, on pourrait créer une vue récapitulative qui permette de visualiser les commandes livrées et les retards dans une seule table. Ce n'est pas le moyen le plus simple, mais cela rend le processus plus lisible.

```
CREATE VIEW retard_commande_livreur AS
SELECT R.CodeLivreur, R.NomLivreur, C.Nombre_Livraison, R.Nombre_Retard
FROM retard_livreur R JOIN commande_livreur C ON R.CodeLivreur = C.CodeLivreur
ORDER BY R.NomLivreur;
```

On distribue les droits de lecture sur cette vue à l'utilisateur 'personnel':

```
GRANT SELECT ON retard_commande_livreur TO personnel;
```

Laboratoire de sociologie. Le laboratoire de recherche doit pouvoir accéder au contenu de deux champs sans calcul ni mise en forme. En revanche, on ne donne pas la possibilité à ces utilisateurs extérieurs de connaître la structure interne des données et d'avoir accès à d'autres champs. On crée une vue qui est le résultat de la jointure entre les tables 'commande', 'client', 'pizza', 'compose', 'ingredient'.

```
CREATE VIEW adresse_ingredient AS
SELECT CL.Adresse, I.NomIngre
FROM commande C JOIN client CL JOIN pizza P JOIN compose CO JOIN ingredient I
ON C.NumClient=CL.NumClient AND C.NomPizza=P.NomPizza
AND P.NomPizza = CO.NomPizza AND CO.NumIngre=I.NumIngre
;
```

On distribue les droits de lecture sur cette vue à l'utilisateur 'sociologue'.

```
GRANT SELECT ON adresse_ingredient TO sociologue;
```

On se trouve dans les cas typiques d'utilisation de vues pour définir des objets qui n'existent pas à l'état « naturel » dans la base de données. La distribution des droits en est d'autant simplifiée. De plus, l'ensemble est évolutif ; on peut sans difficultés ajouter un champ dans ces vues sans avoir besoin de changer l'ensemble des droits sur les tables, les champs, etc.

EXERCICE 6 TRANSACTIONS

Énoncé

On désire effectuer une simulation de mise à jour de la base de données « livraison de pizzas ». L'idée est de procéder à une augmentation générale de 10 % du prix des pizzas, tout en évitant de faire fuir les clients. On essaie de jouer sur les coefficients affectés aux différentes tailles de pizzas (naine, humaine, ogresse) et sur les prix de base. L'objectif est de mesurer les effets de ces modifications sur le compte des clients. On modifie dans un premier temps les prix ; ensuite, les coefficients en se donnant la possibilité de revenir en arrière pour tester différentes combinaisons de coefficients et de prix. Une fois les essais terminés, on abandonne toutes les modifications : ce n'était qu'une simulation.

Solution

On utilise évidemment le mécanisme des transactions en définissant des points de retour pour pouvoir annuler les effets des modifications des coefficients et des prix. On abandonne toutes les modifications, y compris celles des prix à la fin de la transaction.

```
#Démarrage de la transaction
START TRANSACTION ;
# Définition d'un point de retour que l'on nomme 'PRIX'
SAVEPOINT 'PRIX' ;
#Augmentation des prix des pizzas de 10 %
UPDATE pizza SET Prix=Prix*1.1;
```

On a défini ici un point de retour 'PRIX' qui permet de revenir à l'état de la base avant la modification de prix.

```
# Définition d'un point de retour que l'on nomme 'COEFF_NAINE'
SAVEPOINT 'COEFF_NAINE' ;
#Modification des coefficients de la taille 'naine'dans la table tarification
UPDATE tarification SET Coefficient=0.5 WHERE Taille='naine' ;
# Définition d'un point de retour que l'on nomme 'COEFF_OGRE'
SAVEPOINT 'COEFF_OGRE' ;
#Modification des coefficients de la taille 'ogresse'dans la table tarification
UPDATE tarification SET Coefficient=1.5 WHERE Taille='ogresse' ;
```

On a défini ici deux points de retour 'COEFF_NAINE' et 'COEFF_OGRE' qui permettent de revenir à l'état de la base avant les modifications des coefficients.

Pour revenir à l'état de la base avant modification de tous les coefficients, on retourne au point 'COEFF_NAINE'.

```
#Annulation des modifications faites depuis le point 'COEFF_NAINE'
ROLLBACK TO 'COEFF_NAINE';
```

La transaction ne se termine pas lorsque l'on fait un ROLLBACK vers un point de retour. On peut donc essayer d'autres coefficients, sans oublier de redéfinir de nouveaux points de retour.

```
# Définition d'un point de retour que l'on nomme 'COEFF_NAINE2'
SAVEPOINT 'COEFF_NAINE2' ;
#Modification des coefficients de la taille 'naine'dans la table tarification
UPDATE tarification SET Coefficient=0.5 WHERE Taille='naine' ;
```

Et ainsi de suite. Puis, on abandonne l'ensemble des modifications par l'instruction ROLLBACK qui termine la transaction et remet la base dans l'état de départ.

EXERCICE 7 TRIGGER

Énoncé

On désire effectuer des mises à jour « proprement » dans la base de données « livraison de pizzas ». Lorsque l'on détruit une pizza dans la table 'pizza', on veut que les entrées de la table 'compose' correspondant à cette pizza disparaissent automatiquement. Ainsi, la table 'compose'ne contiendra aucune entrée « orpheline ».

Solution

On utilise un trigger (déclencheur) pour mettre à jour automatiquement la table 'compose' lors d'une destruction dans la table 'pizza'.

```
CREATE TRIGGER maj_compose BEFORE DELETE ON pizza FOR EACH ROW
BEGIN
DELETE FROM compose
WHERE NomPizza=OLD.NomPizza ;
END
```

On remarque d'une part que, par rapport aux triggers définis dans le chapitre, on fait appel aux valeurs de la table « OLD ». En effet, lors d'une instruction DELETE, le SGBD ne gère pas de table « NEW » dans la mesure où il n'existe pas de nouvelles valeurs ; dans ce cas, la table « OLD » fera référence aux valeurs supprimées de la table 'pizza'.

D'autre part, on effectue l'opération de destruction dans la table 'compose' avant celle de 'pizza' à l'aide du terme « BEFORE ». Ici, elle aurait peut-être pu intervenir également après, sauf s'il existe une contrainte d'intégrité de référence de la table 'compose' par rapport à la table 'pizza' : un entrée du champ 'NomPizza' dans 'compose' ne peut exister que si elle existe dans la table 'pizza'.

Le trigger permet alors d'automatiser les instructions qu'il aurait été fastidieux d'effectuer « à la main » lors de la destruction d'une entrée dans la table 'pizza'.

Codages de caractères et bases de données

Du code Morse au code ASCII : un rapide historique du codage des caractères

Historiquement, la transmission d'information à distance se faisait de manière visuelle. La marine transmettait des messages entre les bateaux par des **fanions** qui représentaient chacun une lettre de l'alphabet. Le **télégraphe de Chappe** reposait sur le même principe : une position des bras figurait une lettre. Il disposait en outre d'un mécanisme de cryptage ainsi que de codes spéciaux qui accompagnaient le message (par exemple, « urgent », « en attente », etc.). Avec l'invention de l'électricité est arrivé le télégraphe électrique qui utilisait un fil par lettre de l'alphabet (donc, 26 en tout).

Samuel **Morse** invente en 1840 un codage qui passe à un seul fil pour la transmission. Il s'agit du premier codage « moderne » international qui inclut les chiffres, les lettres et certains caractères accentués. Il possède également des caractères de contrôle utilisés pour la transmission du message (par exemple, « répétez »). Ce codage universel permet de transmettre des informations en utilisant une source lumineuse, sonore ou électrique.

Le **télex** apparaît vers 1920 ; il se sert d'un codage sur 5 positions inventé par le Français Baudot. Les messages envoyés par télex sont d'abord saisis sur un ruban perforé et l'envoi ainsi que la réception sont automatisés. C'est le même type de code qui a servi à stocker de l'information sur les cartes perforées à 80 colonnes. Ces dernières représentent le premier stockage d'information avec un code qui préfigure ce qui est utilisé actuellement. Le codage à 5 positions permet 32 possibilités (2^5). Cette capacité est quasiment doublée (diminuée de caractères de contrôle) par l'emploi de deux tables. On indique par un caractère spécifique la table dans laquelle on se situe. Le codage comprend les lettres, les chiffres et quelques caractères de ponctuation. À noter que, dans les caractères de contrôle, on trouve un code pour ramener le chariot de l'imprimante en début de ligne (CR, *Carriage Return*) et un caractère pour passer à la ligne suivante (LF, *Line Feed*).

Les années 1940 voient apparaître les ordinateurs qui recourent au départ aux cartes perforées puis à d'autres systèmes de stockage. Chaque constructeur définit son système de codage ; il arrive parfois que plusieurs systèmes cohabitent chez le même constructeur. IBM définit le codage **BCD** qui servira de base à EBCDIC plus tard et utilise 6 positions.

Dans les années 1960, les constructeurs se sont regroupés pour définir une norme commune qui sera publiée au final par l'organisme de normalisation ASS (*American Standard Association*) sous le nom de code **ASCII** (*American Standard Code for Information Interchange*). Le code ASCII possède 7 positions, ce qui permet de stocker 128 (2^7) caractères. Les 32 premiers caractères sont des caractères de contrôle, un héritage du transfert de données par ruban perforé. Le reste du code contient des caractères majuscules et minuscules, les chiffres et des caractères divers (? @ $, etc.). De nombreuses variantes nationales existent en particulier pour le signe monétaire qui diffère suivant les pays (« $ », pour les USA, « £ », pour la Grande-Bretagne, etc.). En dépit de ces spécificités, l'usage du codage

de caractère ASCII est encore très répandu, car il suffit à bon nombre d'applications à condition que l'on écrive sans lettres accentuées. Si l'on a besoin de caractères diacritiques ou d'autres caractères ne se trouvant pas dans la table de codage ASCII, on peut les représenter par une suite spécifique de caractères ASCII comme on le fait en HTML. Par exemple « é » se représente par la suite « é ».

Tables ISO-8859 et codages propriétaires

Le système de codage des caractères absents de la table ASCII vue plus haut n'est guère commode. Afin qu'il soit possible de représenter d'autres langues, le codage ASCII a été étendu. Le nombre de caractères disponibles a augmenté en passant à un codage à 8 positions, ce qui donne 256 (2^8) possibilités. Les constructeurs ont proposé leurs propres extensions de ASCII. Par exemple, IBM avec « **EBCDIC** » et DEC avec la norme « **VT200** ».

Enfin, l'ISO (Organisation internationale de normalisation) a défini une normalisation de l'extension ASCII pour les langues européennes. L'ennui est que l'on ne peut représenter tous les caractères utilisés dans les langues latines, puisque leur nombre est limité à 256. Plusieurs tables ont donc été définies, le critère choisi pour regrouper les jeux de caractères est fondé sur la proximité géographique et commerciale. On dispose de 16 tables qui seront nommées **ISO-8859-n**, n pouvant varier de 1 à 15. L'avantage est que les 128 premiers caractères sont identiques à ceux de l'ASCII. La compatibilité avec l'existant est donc assurée.

La table définie pour l'Europe occidentale est la table ISO-8859-1, nommée également « **Latin1** », à laquelle il manque évidemment quelques caractères pour être exhaustive. On peut citer l'absence du fameux « œ » français ou du signe « € ». Pour pallier ces manques, une version modifiée de « Latin1 » a été définie : « Latin9 » ou ISO8859-15. Elle est toutefois peu répandue, probablement en raison du temps important qu'a nécessité sa définition. La norme ISO-8859 dispose de tables pour le codage d'autres langues comme le grec (ISO-8859-7), le cyrillique (ISO-8859-5), etc. Les langues extrêmes-orientales utilisent beaucoup plus de 256 caractères et codent les caractères sur plus de 8 positions. Une version allégée du Japonais peut toutefois être codée par une extension de ASCII sur 8 positions : JIS X.

Dans un souci de simplification, les constructeurs ont conçu des extensions de ces normes. En ce qui concerne l'Europe occidentale, on peut citer l'extension de ISO-8859-1, réalisée par Microsoft (**Windows-1252**), qui remplace certains caractères de contrôle de la table « Latin1 » pour y stocker les caractères manquants cités plus haut. Apple utilisait un codage spécifique, « **MacRoman** », incompatible avec les autres.

Remarque

Un point, qui peut se révéler pénible à l'usage, concerne les caractères de fin de ligne. On a vu que le codage, hérité du temps où l'on utilisait des machines mécaniques pour transmettre les caractères, propose deux caractères distincts pour changer de ligne : le retour chariot ('CR') et le passage à la ligne ('LF').
Il n'est plus techniquement utile aujourd'hui de spécifier ces deux caractères pour indiquer une fin de ligne. Cependant, la plupart des protocoles réseaux (par exemple HTTP) ainsi que le monde DOS/Windows ont conservé la combinaison 'CR-LF' pour passer à la ligne. Les familles UNIX utilisent le seul caractère 'LF' et Apple le caractère 'CR'. Il est donc nécessaire d'effectuer des transformations pour passer d'un type de fichier à l'autre.

Unicode

Aujourd'hui, on envoie des mails et l'on s'échange des fichiers d'un bout à l'autre du monde et l'on aimerait pouvoir les exprimer avec les caractères corrects. Le français s'écrit avec des caractères diacritiques : la lecture d'un texte sans accents est désagréable et peut même nuire à la compréhension. La section précédente donne une idée de la confusion qui règne pour le codage des caractères malgré les efforts de normalisation. De plus, on ne peut deviner (c'est-à-dire détecter automatiquement) en ouvrant un fichier s'il est codé en ISO-8859-l ou en MacRoman et il est impossible d'utiliser des tables de langues différentes au sein d'un même fichier. Pour résoudre ces problèmes, un groupement de constructeurs et d'éditeurs de logiciels (Adobe, Xerox, Apple, IBM, MicroSoft, etc.) ont fondé le consortium Unicode au début des années 1990. **Unicode** est un standard défini par un consortium privé, mais après quelques « errements » la norme **ISO-10646** a repris l'ensemble du standard.

L'idée est de disposer d'un système de codage de tous les caractères du monde ou plus exactement de pouvoir coder la représentation de ces caractères. Unicode sépare la notion de caractère de sa représentation que l'on appelle un « **glyphe** ». Ainsi, les notions d'aspect, de police ou de taille du caractère n'existent pas dans le codage Unicode ; ces informations sont reportées au niveau de l'application. Le standard fournit tout de même un exemple de représentation (glyphe) du caractère à titre informatif. En revanche, on inclut des informations complémentaires, comme la direction dans laquelle il faut lire les caractères ou des propriétés alphabétiques qui seront importantes pour les bases de données. En ce qui concerne plus particulièrement les caractères diacritiques, la règle générale est de donner la possibilité de « construire » le caractère plutôt que de le stocker : un « è » sera construit en utilisant le code « e » combiné au code « ` ». Cependant, pour des raisons pratiques de compatibilité, Unicode a repris intégralement les codages existants tels que ISO-8859-1 : dans ce cas, on stocke également le caractère « è ».

La plupart des langues vivantes peuvent s'écrire désormais avec Unicode (plus ou moins 100 000 caractères sont actuellement définis), même si certains choix ont fait l'objet de fortes critiques. Les symboles mathématiques, musicaux ou autres font également partie du code. Le processus se poursuit avec les langues mortes, comme le codage des hiéroglyphes égyptiens.

Techniquement, Unicode utilise un codage sur 21 positions. Le codage est divisé en 17 tables de 65 536 (2^{16}) caractères que l'on appelle des plans. L'un des intérêts de ce découpage est que par exemple le premier plan suffit à coder la plupart des langues vivantes. De surcroît, le tout début de ce premier plan reprend exactement la norme ISO-8851-1. De cette manière, on n'est pas obligé d'utiliser la place des 21 positions pour coder les caractères Unicode. Il suffit de trouver un codage astucieux qui indique que l'on utilise le premier plan ou même le début du premier plan, et alors 8 positions suffisent. Avec des caractères plus « exotiques », le codage utilise alors 16 positions et ainsi du suite. Cet aspect offre beaucoup de souplesse et assure également la compatibilité avec l'existant ASCII et ISO-8859.

Les machines et les logiciels utilisent traditionnellement, et c'est là l'héritage des codages précédents, comme unité de base l'octet qui contient 8 positions. Les différentes manières de coder les caractères Unicode seront donc des multiples d'octets. Essentiellement, on trouve :

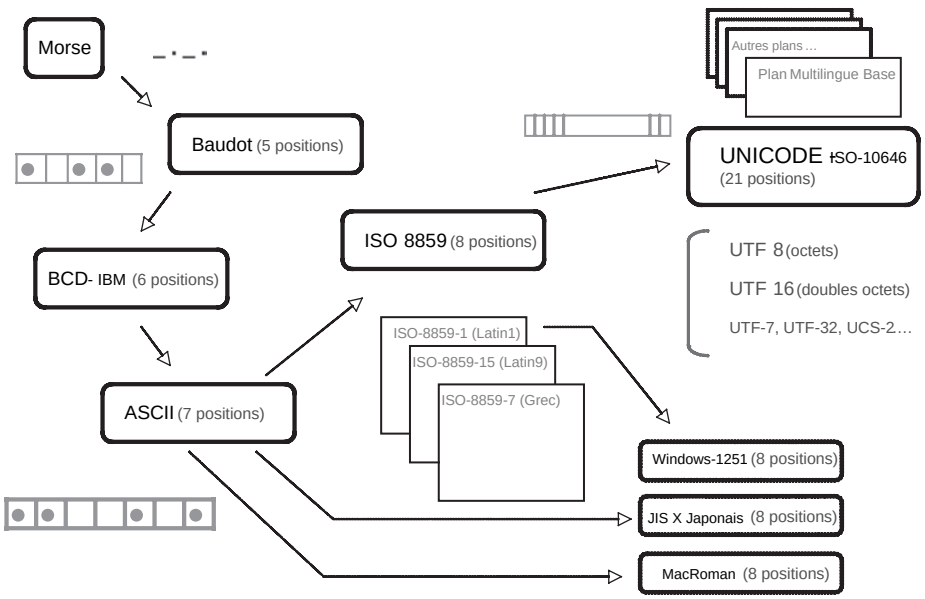
- **UTF-8 :** il s'agit d'un encodage génial sur 8 bits concocté en une journée (une nuit ?) par K. Thompson. L'idée est que tout ce qui est en ASCII est codé sur un octet ; ce qui est codé avec des diacritiques conformes à la norme ISO-8859 est codé sur deux octets et ainsi de suite. Cette manière de coder permet de conserver la compatibilité totale avec le codage ASCII ainsi que de continuer à utiliser les logiciels et matériels conçus à l'origine pour lire des octets.

- **UTF-16 :** on représente Unicode sur deux octets. C'est plus économique en termes de place si l'on utilise fréquemment des langues extrême-orientales (UTF-8 utilise 3 octets dans ce cas). Le problème est de savoir de ces deux octets lequel est le premier (dit de « poids fort »). En effet, deux architectures différentes de machines coexistent toujours, car cette information est codée au niveau matériel. Ces dernières sont désignées par les termes « Little Endian » et « Big Endian ». Il existe donc une version UTF-16LE et une version UTF-16BE. Pour la petite histoire, les termes « Big Endian » et « Little Endian » proviennent du livre *Les Voyages de Gulliver*, où l'auteur décrit des peuplades qui mangent les œufs par le « petit bout », opposées à celles qui les mangent par le « grand bout ».

Il existe d'autres représentations du codage Unicode : UTF-32 sur 32 positions, UTF-7 sur 7 positions, etc. UTF-8 est logiquement le codage le plus fréquemment utilisé actuellement, car il est celui qui consomme le moins de place (si l'on utilise des caractères ASCII ou européens) et qui nécessite le moins de changements structurels aux niveaux logiciel et matériel. À noter que, dans tous les cas, on pourra reconnaître informatiquement le type de codage qui est utilisé. Unicode n'est pas parfait, puisqu'il a intégré les systèmes de codage hérités des systèmes précédents (presque depuis le code Morse…). En revanche, les langues qui ne possédaient aucun codage préliminaire disposent d'un codage élégant et cohérent. Unicode est actuellement supporté par la plupart des systèmes d'exploitation, mais la transformation de tous les textes et bases de données existants prendra du temps : Ken Whistler (directeur technique du consortium Unicode) prévoit environ 10 ans pour la transition.

Figure A.1

Les différents codages de caractères.



Codage dans une base de données et ordre de tri

Les bases de données contiennent des données textuelles. On peut donc trouver dans une base de données tous les codages de caractères présentés plus haut – du simple ASCII à Unicode.

Comment choisir le codage de caractère à employer au moment de la conception de la base de données ?

Une des premières préoccupations est de choisir le codage de caractère qui conviendra aux données entrées. Si l'on utilise les tables de la norme ISO-8859, il faut décider laquelle est convenable en fonction du contenu futur. Le problème ne se pose évidemment pas s'il ne se trouve aucun diacritique dans les données. Une autre question est de savoir ce que les utilisateurs feront des données extraites de la base. Par exemple, sont-ils capables d'utiliser un format de type Unicode ou ont-ils recours exclusivement à Windows-1252 ? De même, disposent-ils de logiciels « clients » capables d'interroger directement la base de données en Unicode ou en IS0-8859-15 par exemple ? Un dernier critère de choix pourrait être la place occupée. Dans le cas du choix d'Unicode, si l'on emploie des langues latines, UTF-8 est plus intéressant que UTF-16. Dans le cas de langues asiatiques, UTF-16 est plus approprié. Le plus économique est sans doute la norme ISO-8859, mais cela est valable à condition que la table utilisée soit suffisamment complète pour les données considérées. Un bon compromis est la norme Unicode codée en UTF-8, qui permet de représenter tous les caractères tout en étant économe en place.

Ordre de tri et collation

Le codage de caractère consiste à associer un code à une lettre. Par exemple on affecte le code 1 à la lettre « a », le code 2 à la lettre « b », et ainsi de suite. Pour trier, il suffit alors de mettre les lettres dans l'ordre des codes. Que se passe-t-il si, dans notre code, la lettre « a » est codée en majuscules et en minuscules ? Si la lettre « A » a le code 30 dans notre codage, le tri par ordre croissant des numéros va se faire d'abord par lettres minuscules puis par lettres majuscules. De même, si notre codage affecte le numéro 50 à la lettre accentuée « à », elle va se retrouver à la suite des majuscules et non pas au même niveau que la lettre « a ». Afin de pouvoir effectuer le tri correctement, il faut disposer d'une table de correspondance qui indique que les codes 1, 30 et 50 de notre codage doivent se trouver au même niveau dans le résultat du tri. On appelle cette table de correspondance la **collation** ; elle sera en général différente suivant la langue employée.

Choix du jeu de caractères et de la collation avec SQL

On doit donc effectuer au moment de la création de la table le choix du codage de caractères à employer ainsi que la collation à lui appliquer.

Le choix du jeu de caractères se fait par le mot clé `CHARACTER SET` suivi du nom du jeu de caractères.

```
Nom_client CHAR(20) CHARACTER SET Unicode ;
```

On peut définir un jeu de caractères spécifique pour chaque champ. Il existe des noms de jeux de caractères prédéfinis dans la norme SQL : `Unicode, ASCII_FULL, LATIN1 etc.` Les SGBD proposent en général des jeux de caractères complémentaires. Il est possible en SQL d'indiquer l'utilisation d'un jeu de caractères prédéfini avec le mot clé `NATIONAL CHARACTER`. En pratique, dans les SGBD actuels, le jeu de caractères employé dans ce cas est le plus souvent UTF-8.

```
Nom_client NATIONAL CHARACTER(20) ;
```

La collation à utiliser est spécifiée par le mot clé `COLLATE` suivi du nom de la collation à utiliser.

```
Nom_client CHAR(20) CHARACTER SET Unicode COLLATE LATIN1;
```

Les SGBD proposent de nombreuses collations en dehors de celles prédéfinies dans la norme SQL. Les jeux de caractères et les collations peuvent être définis plus généralement au niveau des tables, des bases de données et du SGBD lui-même. Un jeu de caractères ainsi qu'une collation par défaut sont utilisés dans le SGBD ; il est intéressant d'en vérifier le contenu avant d'utiliser le SGBD si l'on ne les spécifie pas explicitement.

Bibliographie

G. Gardarin, *Bases de données*, Eyrolles, 2001 (ISBN : 2-212-09283-0).

R. Elmasri et S. Navathe, *Conception et Architecture des bases de données*, Pearson Education, 2004 (ISBN : 2-7440-7055-6).

C. Delobel, C. Lécluse et P. Richard, *Bases de données et Systèmes relationnels*, Dunod.

N. Boudjlida, *Bases de données et Systèmes d'informations. Le Modèle relationnel : langages, systèmes et méthodes*, Dunod, 1999.

C. J. Date, *Introduction aux bases de données*, Vuibert, 2004 (ISBN 2-7117-8640-4).

T. Connoly et C. Begg, *Systèmes de bases de données*, Eyrolles (ISBN 2-89377-267-6).

F. Brouard et Ch. Soutou, *Synthex SQL*, Pearson Education, 2005 (ISBN : 2-7440-7095-5).

B. Charroux, A. Osmani et Y. Thierry-Mieg, *Synthex UML2*, Pearson Education, 2005 (ISBN : 2-7440-7124-2).

Ch. Soutou, *De UML à SQL*, Eyrolles, 2002 (ISBN : 2-212-11098-7).

H. Tardieu, A. Rochfeld et R. Colleti, *La Méthode MERISE*, Eyrolles, 2000.

D. Nanci et B. Espinasse, *Ingénierie des systèmes d'information : Merise deuxième génération*, Vuibert, 2001 (ISBN 2-7117-8674-9).

P. André et A. Vailly, *Conception des systèmes d'information, panorama des méthodes et des techniques*, Technosup, Ellipse, 2001 (ISBN 2-7298-0479-X).

E.F. Codd, *A Relational Model of Data for Large Shared Data Banks*, 1970.

Index

L

M

N

O

Q

R

S

T

U

V

W

X

Informatique

Synthèse de cours & exercices corrigés

Création de bases de données

L'auteur :

Nicolas Larrousse est ingénieur au CNRS. Spécialisé en informatique, il enseigne les bases de données à l'université de Versailles Saint-Quentin-en-Yvelines et au service de formation permanente de l'université Pierre et Marie Curie à Jussieu.

Le relecteur :

Éric Innocenti est maître de conférences en informatique à l'université de Corse. Il est responsable pédagogique des filières SRC (Services et Réseaux de Communication) et LPM (Licence Professionnelle Multimédia). Il enseigne l'algorithmique, la programmation et les systèmes d'information.

Dans la même collection :

- **Algorithmique, Applications en C**, Jean-Michel Léry
- **Algorithmique en C++**, Jean-Michel Léry
- **Algorithmique en Java 5**, Jean-Michel Léry
- **Architecture de l'ordinateur**, Emmanuel Lazard
- **Java 5**, Robert Chevallier
- **LateX**, Jean-Côme Charpentier, Denis Bitouzé
- **Le langage C**, Jean-Michel Léry
- **Le langage C++**, Marius Vasiliu
- **Linux**, Jean-Michel Léry
- **Mathématiques discrètes appliquées à l'informatique**, Rod Haggarty
- **SQL**, Frédéric Brouard, Christian Soutou
- **Systèmes d'exploitation**, Bart Lamiroy, Laurent Najman, Hugues Talbot
- **Réseaux**, Dominique Seret, Danièle Dromard
- **UML 2**, Benoît Charroux, Aomar Osmani et Yann Thierry-Mieg

Cet ouvrage propose une démarche progressive à ceux qui veulent concevoir un système d'information robuste et évolutif en évitant les écueils classiques qui conduisent à rendre les données inutilisables. Toutes les étapes de la réalisation d'une base de données, de l'analyse préalable au choix du codage des caractères, sont étudiées et illustrées par des exemples.

Le livre présente plus particulièrement la modélisation du monde réel au moyen du modèle entité-association, le passage au modèle relationnel et la mise en œuvre du système ainsi conçu à l'aide du langage SQL. Une étude de cas récapitulative permet ensuite d'appliquer les notions présentées dans les chapitres précédents. Le dernier chapitre traite de la sécurisation des données, notamment au moyen des transactions et des triggers.

Les exercices, qui occupent la moitié du livre, sont intégralement corrigés afin que le lecteur mette progressivement en œuvre ses connaissances. Par ailleurs, les données et les scripts SQL utilisés tant pour les exemples que pour les exercices sont disponibles à l'adresse www.pearsoneducation.fr.

Le livre s'adresse aux étudiants de premier et de second cycles (IUT, BTS, universités et écoles d'ingénieurs) qui débutent l'apprentissage des bases de données. Il sera également utile aux professionnels qui veulent mettre en place une base de données, même de taille modeste.

La collection *Synthex informatique* propose de (re)-découvrir les fondements théoriques et les applications pratiques des principales disciplines de science informatique. À partir d'une synthèse de cours rigoureuse et d'exercices aux corrigés détaillés, le lecteur, étudiant ou professionnel, est conduit au cœur de la discipline, et acquiert une compréhension rapide et un raisonnement solide.

PEARSON

Pearson Education France
47 bis, rue des Vinaigriers
75010 Paris
Tél. : 01 72 74 90 00
Fax : 01 42 05 22 17
www.pearson.fr

ISBN : 978-2-7440-7386-1